CONTINUOUS PLANNING AND EXECUTION WITH TIMELINES

Amedeo Cesta[1], **Simone Fratini**[2], **Andrea Orlandini**[1], **and Riccardo Rasconi**[1]

[1]*CNR – Consiglio Nazionale delle Ricerche, ISTC, Rome, Italy – name.surname@istc.cnr.it*
[2]*ESOC-ESA – European Space Agency, Darmstadt, Germany – simone.fratini@esa.int*

ABSTRACT

Planning systems need to be endowed with some additional features to cope effectively with execution: e.g., the ability to keep the plan database updated with respect to the actual feedbacks provided by the controlled system, to mention but one. In this paper, we identify a set of noteworthy planning and execution open issues relatively to the timeline-based planning approach. We address those issues presenting a domain independent deliberative system, implemented on top of the APSI-TRF, the APSI Timeline-based Representation Framework, extended with timeline dispatching and execution-supervision capabilities so as to allow continuous planning and closed-loop re-planning activities. Some ongoing research directions are also briefly introduced.

Key words: Timeline-based Planning and Execution, Model-based Control, Planning and Scheduling.

1. INTRODUCTION

Deep space and remote planetary exploration missions are characterized by severely constrained communication links, limited in communication window durations and data transmission rates. Then, controlling remote spacecrafts for reliably conducting scientific and engineering operations represents a challenging issue. This is not only because fast reaction is sometimes needed, but also because, without access to live data, decisions made remotely by human operators may be based on obsolete information, which could be not suitable and even hazardous to the system. Planning and execution systems constitute a suitable solution, but the ability of dealing with time uncertainty and to reason over metric time and resources is required. Then, deliberation is intrinsic, but it also requires monitoring and immediate reactive response to evolving conditions that a remote system has to face.

The GOAC [5] project was an ESA initiative aimed at defining a new generation of autonomous controllers to support increasing levels of autonomy for exploration space robotic tasks achievement. In particular, goals of the GOAC architecture are to: generate on-board activity plans, monitor and dispatch activities during execution, and recover from off-nominal conditions. The pursued GOAC solution has been designed as a principled integration of different software solutions: (1) a timeline-based deliberative layer implemented on top of the APSI Timeline-based Representation Framework (APSI-TRF) [11], (2) an executive layer based on T-REX [23] devised at MBARI, and (3) a functional layer which integrates the G^{en}oM and BIP systems devised at LAAS and VERIMAG, respectively [4].

Within the GOAC architecture, a functional layer is tightly integrated with a decisional layer, both of which have a rich history with deployments in complex, real-world environments. The decisional layer is usually constituted by a hierarchy of deliberative reactors where higher levels deal with long-term mission plans while lower levels are increasingly reactive. The functional layer is purely reactive with fast reaction times necessary for failure recovery and low-level command dispatching. Additionally, a verification tool ensures a correct-by-construction functional layer implementation, with respect to properties such as deadlock-freedom.

In this work, we identify a set of noteworthy open issues in planning and execution systems relatively to the timeline-based planning approach: the dynamic management of goals during planning and execution, the assessment of the status of partially executed goals and the dynamic dispatching of commands. Then, we describe a timeline-based, domain independent deliberative control system, called the "APSI Deliberative Reactor"[1], implemented within the GOAC project and addressing the considered open issues. In particular, we present a proactive control system entirely based on the ESA APSI-TRF technology. The APSI Deliberative Reactor is constituted by (i) an execution module, to dispatch planned timelines, to supervise their execution status and to entail continuous planning and re-planning, (ii) a new timeline-based planning module, called the "AP2 planner", to model and solve planning problems.

The APSI Deliberative Reactor is devised to be domain independent, i.e., providing a suitable timeline-based description model of the system to be controlled and a set of temporal goals to be achieved, it fully implements all the functionalities required to plan for goals, to dispatch planned values to the controlled system and to supervise plan execution ingesting the telemetry of the controlled system. One of the main advantage of being domain independent entails both the capability of the deliberative reactor to plan for user goals and to dynamically react to

[1]The term *Reactor* is a legacy from T-REX. It is worth also saying that the initial motivation of our work is to design a smooth integration with the T-REX executive that in its original implementation uses a different timeline-based planner.

off-nominal conditions detected from the controlled system telemetry, as well as a twofold flexibility: in being used to control different systems (by substituting the controlled system domain description) and in the capability of achieving different classes of user goals for the same system (by substituting the controller model).

Additionally, it is worth point out that such approach allows to implement the controller system as a hierarchical composition of a set of deliberative reactors to refine high levels goals into more detailed low level plans. In this case, the higher level reactors generate goals for the lower level reactors as well as telemetry from lower levels reactors represents plan achievement information for higher levels reactors. In the GOAC project, the executive layer, based on the T-REX architecture, uses such a hierarchical configuration of reactors (see [5] for more details).

The paper is structured as follows: Section 2 discusses the related works while in Section 3, a set of planning and execution open issues are discussed. Then, in Section 4, the APSI Deliberative Reactor is presented as well as its capabilities are discussed. Section 5 presents the deployment of the APSI-TRF in the GOAC scenario. Finally, Section 6 briefly presents current research efforts aiming at further extending the APSI-TRF and its capabilities.

2. RELATED WORKS

The dominant approach for building control agent systems utilizes a three-layered architecture [14]. Most of the traditional autonomous control architectures follow this approach, and they mostly differ in the choice of which layer dominates over the others.Some of the most relevant works on three-layered control concerns: IPEM [2], CPEF [20], the LAAS Architecture [1] (exploiting the IxTeT-eXeC [18]), the Remote Agent Experiment [16] and ASE [12]. Each layer (i.e., *Functional*, *Executive* and *Planning/Scheduling*), generally requires different reasoning technologies and representations, and their overall integration is usually difficult to achieve and test. Indeed, many of these systems utilize very different techniques for specifying each layer in the architecture, which results in an unnecessary duplication of both efforts and information. As a further example, the fact that the planning cycle is often monolithic makes scalability a great concern indeed; with these approaches, obtaining fast reaction times when necessary is often impractical.

More recent approaches, such as CLARAty [21], try to overcome some of the these drawbacks using an architecture with two layers: a *functional* layer and a *decision* layer. The decision layer integrates planning and execution through a tightly-coupled database that synchronizes planning and execution data from two different representations: one from CASPER [17] (planning) and the other from TDL [24] (execution). Another two-layer architecture is the model-based approach pursued by Williams [25]. This framework consists of a Reactive Model-Based Programming Language (RMPL) and an executive (Titan). RMPL is a rich language which provides a framework for constraint-based modeling, as well as a suite of reactive programming constructs. RMPL abstracts the state of the functional level so the programmer can reason in terms of state variables not directly corresponding to observable or controllable states. Anyway, it is not clear in this approach how the underlying cost of diagnosis/planning can be controlled. Finally, in [15], the automatic controller synthesis has been addressed exploiting reactive planning in order to implement intelligent control in hard real-time. In particular, CIRCA incorporates model-checking into the control architecture, does abstraction-based controller synthesis, and extends the modeling capabilities for hierarchical control.

IDEA [19] was the first agent control architecture utilizing a collection of controllers, each interleaving planning and execution in a common framework. Main drawbacks of IDEA are the lack of: (1) a clear conflict-resolving policy between controllers, and of (2) an efficient (e.g., non-exponential) planning algorithm for integrating the controller's current state. In order to make the approach effective in practical cases, an efficient state synchronization algorithm for partitioned structures is fundamental. The Teleo-Reactive Executive (T-REX) [23] was designed to overcome these restrictions using a collection of controllers (*reactors*) implemented as different instances of the EUROPA$_2$ planner [13]. The novelty of T-REX relies in its capability to enforce a systematic infrastructure to define the interactions between the reactors. Then, providing a formal framework for both state-synchronization and plan-dispatching critical primitives, T-REX ensures a correct behavior of the agent and makes the approach efficient and scalable in practice.

3. PLANNING AND EXECUTION ISSUES

Generally speaking, a planning and execution system should embrace the philosophy that plans are dynamic, open-ended artifacts that must evolve in response to an ever-changing environment. In particular, plans and activities are to be updated in response to new information and requirements to ensure that they remain viable and relevant [20]. In this regard, a planning and execution system should perform several tasks in order to guarantee not only that a safe and correct plan can solve the given planning problem, but that the system is able also to actually execute the plan as well as monitoring its execution.

The T-REX system allows to (i) enforce the synchronization of the current solution plan database with observations dynamically gathered from the actual system, (ii) dynamically generate plans for user goals and (iii) dispatch commands accordingly with its internal status. Nevertheless, some aspects are still far to be addressed. Namely, the following issues are still open:

Dynamic management of goals. As the planning and execution system should enable continuous (re)planning in a dynamically evolving situation, the system should be able to manage the dynamic flow of incoming goals. In fact, additional goals can be sent to the control system while executing already generated plans. Then, the system should be able to dynamically manage goals in an effective way enforcing the agent capability of fulfilling all the goals as well as ensuring the overall plan stability. Moreover, once an inconsistency is detected, the system

should be able to build a new safe initial state and then ask the planner to provide a new plan for the original user goals (if still reachable). Then, an assessment is required in order to decide which goals are to be maintained and which goals are to be discharged.

Assessing goals execution status before recalling. Strictly related to the above aspect, another important point concerns the system capability of recalling the (pending) commands not yet executed. Indeed, evaluating whether a goal is still pending is not a trivial issue. In fact, it is not easy to assess which is the status of a dispatched command. That is, can be recalled a goal that is actually in execution? Can be recalled a goal that is related with some sub-goals that have been already executed? Thus, the system should be able to decide which is the actual execution status of a goal in order to take the more proper decision on goal recall.

Dynamic timelines execution. Once a planner has generated a temporal plan, it is up to the executive system to decide, at run-time, how and when to execute each planned activity preserving both plan consistency and controllability. Such a capability is even more crucial when the generated plan is temporally flexible being a flexible temporal plan partially specified. Such a plan captures an envelope of potential behaviors to be instantiated during the execution taking into account temporal/causal constraints and controllable/uncontrollable activities and events.

4. PLANNING AND EXECUTION WITH THE APSI FRAMEWORK

Design and implementation of advanced Planning and Scheduling software for space applications is an expensive activity involving a certain amount of developing effort and risk. For instance, the software may fail to meet operational requirements (performance issues), and/or may fail to capture all the essential aspects of the problem (modeling issues).

With the aim of improving the process of tool design and deployment, taking advantage of state of the art planning and scheduling technology, an ESA initiative, spanning [2007-2009], led to the development of a software platform, called the APSI Timeline Representation Framework, shortly APSI-TRF. The aim of the framework is to support developers and engineers to cope with both software deployment efforts and modeling risks mentioned above. The APSI-TRF simplifies the design and developing effort by providing a library of timeline-based planning and scheduling domain independent solvers and enhancing the robustness of the interactions among the specific solvers implemented on top of the framework by providing a uniform representation of the solution database. Modeling risks are reduced because the use of a framework that standardizes and simplifies the process of application deployment fosters a rapid prototyping cycle, which directly helps the users to take into account their own feedback during the application design and because common Domain and Problem definition languages are available and reusable with all the applications deployed within the framework.

In order to implement an effective planning and execution system as well as to cope with the open issues discussed in the previous section, a new intelligent components has been defined by means of two different modules designed and implemented on top of the APSI-TRF: a planner to generate flexible plans, namely the AP2 planner, and a module to entail and supervise the execution of planned timelines, namely the APSI Deliberative Reactor. The architecture of the interactions between these modules is shown in figure 1.

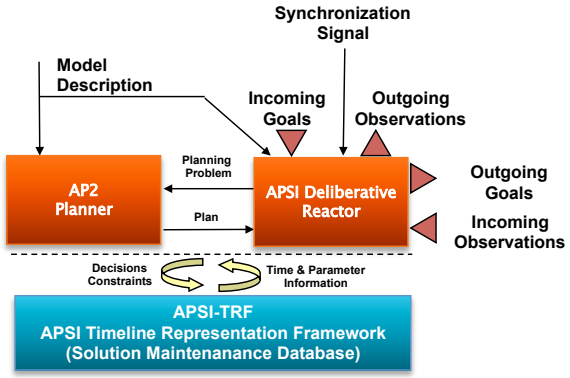

Figure 1. Architecture.

In this architecture, the APSI-TRF plays the role of the solution maintenance database, the AP2 planner is the main solver in charge of producing timelines to achieve goals and the deliberative reactor, that interacts with the APSI-TRF to supervise the execution of the plans. The reactor is in charge of three main tasks: enforcing the synchronization of the solution database with observations in input, using the planner for planning and re-planning and dispatching commands accordingly with its internal status. The planner is invoked by the deliberative reactor that provides problems, either to achieve user goals (planning) or to fix the plan in execution in case of need (replanning), and collect solutions to dispatch. In order to address the issues presented in Section 3, the Deliberative reactor is endowed with four ad-hoc policies: (i) a Goal Planning Policy, to dynamically manage goals; (ii) a Goal Achievement Policy, to assess the status of pending goals according to their actual execution status; (iii a Goal Replanning Policy, to decide which goals are to be maintained for the replanning phase; (iv) a Goal Dispatching Policy, to execute plans ensuring plan consistency and controllability.

The AP2 generates *time flexible* plans as a set of timelines where the transitions between values are not fixed but temporally related with a set of constraints. Each transition as well as temporal relations with other timelines are defined with lower and upper bound for the planned time. These planned relations must be achieved to guarantee the validity of the plan. Any transition out of planned bounds or violating the relations is not valid and a plan execution failure must be taken into account. Timelines and constraints are stored in the solution database which is, must be taken into account, a dynamically evolving database because concurrently with the planner the reactor modifies the database during execution to synchronize the status of the database with the status of the real world (this process is detailed below).

In this configuration, the planning process take place

within a bounded time window and must be not blocking for other modules that interact with the solution database (like the reactor). Planning in a bounded window is needed to maintain the stability of the part of the plan currently in execution while the planner's solving process is implemented as a sequence of very short reasoning bursts (a few milliseconds each), so that it can be interrupted with no side effects on the portion of the timelines in execution (should synchronization problems occur that suddenly require the planner for more urgent duties). This setting allows the use of the planner without interrupting concurrent reactor operations.

The reactor module is based on the the principle that the timelines that constitute the domain are *functionally* partitioned into *internal* and *external*. The functional distinction between internal and external timelines aims at distinguishing between the timelines that describe the controller internal status and the timelines that describe the status of the system that has to be controlled (basically the distinction between the controller model and the controlled system model). This distinction is important, as the internal status of the controller can be decided by the planner (i.e., it will surely be reached as expected), while the status of the controlled system can only be planned for by the reactor but there is no guarantee that it will be actually reached.

Resembling the main T-REX algorithm, the reactor implements a control loop based on the following activities: (i) receiving the current status of the controlled system through a series of telemetry values (i.e., actual values of external timelines, a.k.a. *observations*); (ii) attempting to synchronize its own view of these timelines in the plan database (believed status of the controlled system) with the *observed* timeline values; interleaved with this activity; (iii) the reactor plans for the desired status of the controlled system required to achieve user goals; (iv) dispatches this status by means of *goals* to be achieved for external timelines (under the control of other reactors/planners). The actual achievement of these values is supervised by means of the *synchronization* process.

Three main modules are in charge of the reactor functions (see Figure 2): a *core* module supervises the process, an *observer* module synchronizes incoming observations and a *dispatcher* module dispatches outgoing goals. A fourth module, i.e., the AP2 *planner*, is needed to perform planning for user goals [2].

The core module coordinates the other modules while managing planning processes and synchronization procedures. The core module implements the user goal life cycle shown in Figure 3.

With reference to the labels in Figure 2, the goal life cycle is implemented according to the following steps: **(G1)** incoming user goals are collected and buffered into a *goal manager* module (the goal is BUFFERED); a "Goal Planning Policy" (GPP) implemented in the goal manager module controls the progression of the goal from the status BUFFERED to the status IN PLANNING. The actual implemented GPP allows either a first-come-first-served

[2] In the GOAC project the used planner AP2, but this module can in principle use any APSI-based solver.

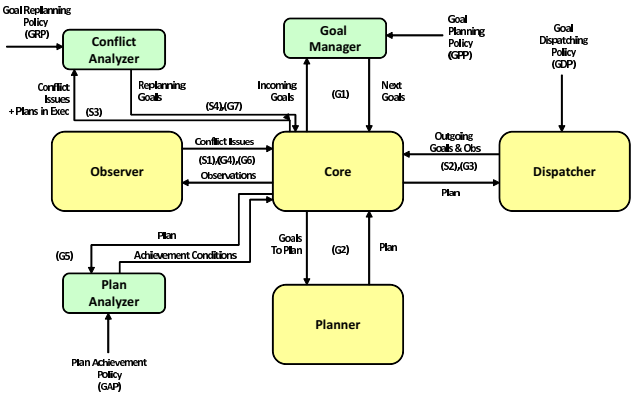

Figure 2. Reactor Architecture

approach, or an early start time schedule approach or a user-specified policy **(G2)** the goals selected by the GPP to be planned next are passed on to the planner. Once the time allotted for planning has expired, the transition from IN PLANNING to PLANNED is triggered. **(G3)** Once planned, the produced plan is passed on to the Dispatcher module. A "Goal Dispatching Policy" (GDP) implemented by the dispatcher controls the progression of the goal from PLANNED to DISPATCHED. Two policies are implemented: the first one dispatches all goals as earliest as possible (basically as soon as the plan is available), the other one dispatches goals as latest as possible (when the time frame for the goal is approaching). The first approach gives to other reactors more time to plan for goals, but is more heavy in case of failures because goals have to be recalled, while the latter approach is more risky because goals are visible to other reactors only close to their time frame but in case of failures they do not need to be recalled. Once selected to be dispatched by the GDP, the goal is sent to the controlled system. **(G4)** the goals come in execution as soon as any of the values that constitute the plan gets in execution. The Observer module, in charge of supervising the execution of the reactor timelines, triggers the transition from DISPATCHED to IN EXECUTION. **(G5)** once in execution, a "Goal Achievement Policy" (GAP) controls the progression from IN EXECUTION to EXECUTED. A *plan analyzer* module implements the GAP. The module analyzes the plan and produces a set of *goal achievement conditions* that the *core* uses to assess the correct execution of the plan. Currently there are two goal achievement conditions implemented: "at start" (the goal is achieved when the selected value appears on the timelines) and "at completion" (the goal is executed when the selected value dis-

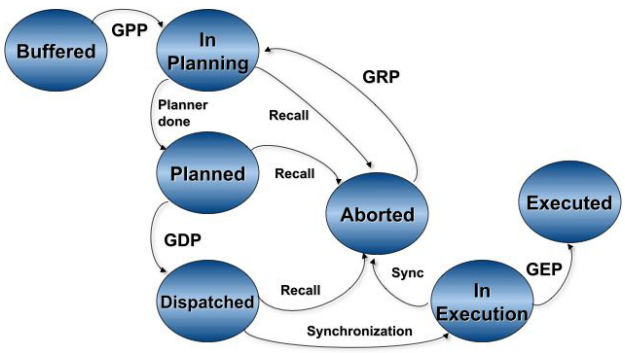

Figure 3. User Goal Life Cycle.

appear from the timelines). Whether a given value model something that is considered achieved at start (like a goal "At(x,y)" for instance to move the platform in (x,y) that can be considered achieved as soon as the value appears) or at completion (like a goal "TakePicture" for instance that can be considered achieved only when the process is finished and the timeline change value) is a piece of knowledge that only the modeler knows, hence they are specified in the domain model via tags and used by the reactor to correctly assess when a given goal can be considered achieved. **(G6)** The Observer module, while synchronizing timelines, can force the transition of the goal from the status IN EXECUTION to ABORTED as soon as any synchronization problem arises (an incorrect execution of any of the values of the plan is detected). In such case, **(G7)** a "Goal Re-Planning Policy" (GRP) controls the possible transition of the goal status from ABORTED to IN PLANNING again. A *conflict analyzer* modules implements the GRP. The GRP evaluates the situation on the basis of a set of *conflict issues* generated by the Observer module, and can decide if and when to re-plan for the goal. Currently there are the following general policies implemented: re-schedule (the goal(s) currently in execution when the failure occurs are re-scheduled in input as newly posted goal(s)), re-plan (the goal(s) are re-planned immediately no matter of other goals eventually in the buffer), skip (the goal(s) are discarded) or user-oriented. In the latter case the user can specify which goals have to be passed to the planner in order of to react to the failure. When the goal is BUFFERED, IN PLANNING, PLANNED and DISPATCHED, it can be *recalled* forcing a transition into a ABORTED status. The current implementation of the reactor does not allow to recall a goal once the plan for that goal is in execution.

The goal life cycle management process is interleaved with the timeline synchronization process. Again with reference to the labels in figure 2, the following steps constitute the reactor synchronization procedure: **(S1)** incoming observations that describe the actual status of the controlled system are passed on to the Observer module. **(S2)** if no synchronization problem arises, the observed status of the controlled system is compatible with the current plan database; the plan database is updated and the out going controls are produced. **(S3)** if a synchronization problem arises, the current status of both the controller and controlled system is re-calculated, and a list of *conflict issues* is generated for the *Conflict Analyzer*. **(S4)** On the basis of the implemented GRP, the module provides the core with a set of goals to re-plan for (starting from the re-calculated status). Conflicts are detected by the observer module when (1) an incoming observation puts an unexpected value on a timeline forcing an invalid state transition (the controlled system is taking an unexpected status), (2) an incoming observation forces a transition on a state that cannot be changed (the controlled system truncates an operation before the planned time) or (3) when there is no incoming observation but the current status cannot be maintained (the controlled system is not blocking an operation at the planned time).

The reactor is designed to be modular and easily extensible with respect to the implemented policies. This features makes the behavior of the reactor easily customizable for different domains. In fact, especially for what the goal planning and re-planning policy (GPP and GRP) are concerned, it is very difficult to design scenario independent policies that are general and efficient at the same time. Moreover the policies can change also for different runs of the same scenario, to achieve different user needs. Hence a design choice of bet on modularity more than in implementing fancy embedded policies. The next section shows an illustrative scenario with an example of problem dependent policies.

5. AN ILLUSTRATIVE SCENARIO

This section describes an illustrative scenario related to the GOAC project. First, we describe the DALA platform [3], i.e., the real robotic platform deployed within the GOAC project. Then, we exploit this scenario in order to show a possible configuration of the control system.

5.1. The Robotic Platform

DALA is one of the LAAS-CNRS robotic platforms that can be used for autonomous exploration type of experiments. In particular, it is an iRobot ATRV robot that provides a large number of sensors and effectors. It can use vision based navigation (such as the one used by the Mars Exploration Rovers Spirit and Opportunity), as well as indoor navigation based on a Sick laser range finder. In particular, the purpose of the scenario is basically to simulate as close to a Mars or Lunar rover as possible.

DALA can be considered as a fair representative for a planetary rover equipped with a Pan-Tilt Unit (PTU), two stereo cameras (mounted on top of the PTU), a panoramic camera and a communication facility. The rover is able to autonomously navigate the environment, move the PTU, take high-resolution pictures and communicate images to a Remote Orbiter. Moreover, the rover is also able to monitor the environment with a panoramic camera for possibly interesting scientific features. Finally, during the mission, the Orbiter may be not visible for some periods. Thus, the robotic platform can communicate only when the Orbiter is visible.

It has a mission to take high-resolution pictures of different locations with an associated PTU configuration, which have been requested by scientists. Then, a possible mission action sequence is the following: navigate (using a navigation mode compatible with the type of terrain) to one of the requested locations, move the PTU pointing at the requested direction, take a picture, then, communicate the image to the orbiter during the next available visibility window, put back the PTU in the safe position and, finally, move to the following requested location. While navigating, the rover may be requested to check for unforeseen interesting scientific features. Then, when such a feature is detected, the robot takes an opportunistic science picture and send this information to the orbiter. Once all the locations have been visited and all the pic-

[3]Thanks to Felix Ingrand and Lavindra De Silva from LAAS-CNRS for the time spent to explain us the details of their robotic platform.

tures have been communicated, the mission is considered successfully completed. See [5] for more details.

5.2. Control System Configuration

Following [23], a control system configuration has been designed considering an analogy in human control for the mentioned rover, then, implementing a suitable control system as the composition of a set of deliberative reactors. In this regard, some different personnel acting specific roles can be considered as involved in control tasks:

- A *Mission Manager* controls the general mission plan via high-level goals such as where to go and which areas to inspect looking for interesting objects.
- A *Science Engineer* has the control over the scientific needs providing detailed plans for both scientific pictures and environment monitoring activities.
- A *Platform Engineer* satisfies the rover configuration requests controlling the mobility system and the PTU while respecting operational constraints as well as commanding the rover for a successful mission completion.
- Finally, a *Command Dispatcher* receives and executes the commands requested in order to activate the functional level behaviors.

Thus, each reactor has a specific functional role over different temporal scopes during the mission. For instance, while the Platform may need to react immediately to a potential navigation problem, the Mission Manager reactor may need time to deliberate in order to alter mission objectives in the light of new information. Indeed, the reactors represent sense-plan-act loops with their own scopes and view of the world, while interacting with each other in order to achieve the global mission objectives. In particular, the Mission Manager's temporal scope is the entire mission and potentially it can take minutes to deliberate. The Command Dispatcher on the other hand interfaces to the DALA functional layer and needs to have minimal latency with no deliberation. Finally, the Science and Platform reactors have temporal scopes in between.

More in detail, the Mission Manager reactor is designed to provide plans for user requested goals, i.e., requests for (i) scientific pictures in desired locations, (ii) reaching a certain position and (iii) monitoring a certain area. Then, the timelines planned by the Mission Managers are dispatched for execution to the Science reactor, in order to perform scientific photo or to monitor a certain area. In particular, once a photo has been taken, a communication request is produced for the command dispatcher. On the other hand, the request of reaching particular locations is directly dispatched to the Platform reactor. The Science reactor is designed to encode take picture and monitoring requests, in synchronizations of robot movements while looking for additional science opportunities (possibly triggered by the monitoring task). The Science reactor is to ask the Platform to reach a particular configuration (location and PTU configuration) in order to take a requested picture as well as implementing specific low-level goals for the Command Dispatcher to shoot photos and perform science monitoring. The Platform reactor is designed to provide navigation independent from the terrain, enforce PTU positions for robot moves and hide terrain switch events The produced low-level goals are to control the mobility system and the PTU of DALA. The Command Dispatcher encodes the planned values into actual commands while producing observations for the low-level timelines relying on the replies provided by the functional layer.

6. CURRENT WORKS

Currently, new research efforts are aiming at enriching the APSI-TRF, and the GOAC deliberative functionalities, with additional capabilities investigating different relevant aspects.

A research line is investigating the exploitation of Validation and Verification techniques. In particular, the KEEN environment [6] is composed of different V&V modules implementing design support functionalities. A Timed Game Automata (TGA) Encoding module is deputed to implement a translation from P&S specification to TGA. The encoding method has been presented in [7] and shares the same formal results presented in [8, 9]. Then, relying on that encoding and results, the KEEN system is endowed with: a Domain Validation module, to support the model building activity providing a tool to assess models quality; a Planner Validation module, to assess the P&S solver with respect to system requirements; a Plan Execution Validation module, to check whether the proposed solution plans are suitable for actual execution or not. In order to implement the modules functionalities, the verification tasks are performed by means of UPPAAL-TIGA [3], a toolbox for the specification, simulation, and verification of real-time games. Recently, the KEEN environment has been endowed with a new functionality: a method to synthesize robust plan controllers for timeline-based flexible plans solving a TGA model checking problem [22, 10].

Still related to the GOAC project, another initiative just about to start and, then, currently under development is the creation of an On-Ground Autonomy Test Environment (OGATE) [4]. This initiative is intended to synthesize an integrated environment to test features of autonomy software of different goal oriented controllers and to explore quantitative comparisons based on accurate experiments and also qualitative analysis allowed by inspection and visualization of software internal monitors and a simulation-based verification. Broadly speaking, the system will pursue the possibility of performing comparative tests of different architectures for deliberative modules in autonomy software. Additionally, the OGATE will also investigate the possibility to implement mixed decision making between human-operators and autonomy

[4]This is a 2012 ESA Networking/Partnering Initiative among ESA, University of Alcalá and CNR.

software (e.g., test of different policies of goal specification).

Finally, a multi-agent extension of the GOAC architecture is under investigation. In fact, in domains such as space exploration, it is worth studying scenarios with multiple robots working together in order to increase efficiency in terms of goals completed. In addition, single-agent architectures do not offer the necessary fault tolerance that multiple-agent coordination may offer. The envisaged scenario is such that each goal can be carried out by a team composed by one or more robotic agents. Agents develop plans to carry out goals using timelines as well as operate in unknown environments where failures occur either due to exogenous conditions, or due to internal failures. The problem is that of optimizing efficiency, such that agents accomplish more goals. The presence of human agents will be also considered in order to implement a mixed-initiative multi-agent system where robots and humans collaborate in order to accomplish the mission.

7. CONCLUSIONS

This paper describes a timeline-based, domain independent deliberative layer (based on the APSI-TRF technology) to be used in the Goal Oriented Autonomous Controller (GOAC) software environment. The control system is designed to be domain independent, i.e., it takes in input a (timeline-based) model of the system to be controlled, a (timeline-based) model of the specific controller and a set of user goal to be (hopefully) achieved by the controlled system. The controller uses a planner to plans for user goals, dispatches commands for the controlled system (the status planned for the controlled system in order to achieve the user goals) and supervises plan execution by reasoning on the telemetry of the controlled system (the actual status of the controlled system). Some open issues related to planning and execution with timelines have been discussed and addressed.

In particular, the proposed control system is constituted by:

- a new planning module, referred to as the "AP2 planner" that exploits the timeline-based approach provided by the APSI-TRF to model and solve timeline-based planning problems. The AP2 planner generates time flexible plans: a set of timelines where the transitions between the values are not fixed by the planner but temporally related with a set of constraints. The flexibility of the produced plans and the capability of re-planning entail the suitability of the system in controlling physical systems, once coupled with an executive layer that takes care of properly dispatching the plan and supervise its execution.
- a module, called the "APSI Deliberative Reactor", that implements the *sense-plan-act* cycle as well as the goal life-cycle management. The module dispatches planned timelines and supervises their execution status to enforce continuous synchronization with the status of the controlled system and to allow planning and re-planning.

The advantage of the controller being based on a domain independent planner entails both the capability of the controller to plan for goal and to dynamically react to off-nominal conditions detected from the controlled system telemetry, as well as a twofold flexibility: in being used to control different systems (by substituting the controlled system domain description) and in the capability of achieving different classes of user goals for the same system (by substituting the controller model).

Acknowledgment. Authors were partially supported by the ESA GOAC project (TRP/T313/006MM). Amedeo Cesta and Riccardo Rasconi are currently partially supported by MIUR under the PRIN project 20089M932N (funds 2008). Andrea Orlandini has been supported by a grant within Accordo di Programma Quadro CNR-Regione Lombardia: Progetto 3.

REFERENCES

1. R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *International Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming*, 17(4):315–337, April 1998.
2. J. Ambros-Ingerson and S. Steel. *Integrating Planning, Execution and Monitoring*, volume 1, pages 83–88. AAAI Press, 1988.
3. G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime. UPPAAL-TIGA: Time for playing games! In *Proc. of CAV-07*, number 4590 in LNCS, pages 121–125. Springer, 2007.
4. S. Bensalem, L. de Silva, M. Gallien, F. Ingrand, and R. Yan. "Rock Solid" Software: A Verifiable and Correct-by-Construction Controller for Rover and Spacecraft Functional Levels. In *i-SAIRAS-10. Proc. of the 10^{th} Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*, 2010.
5. A. Ceballos, S. Bensalem, A. Cesta, L. de Silva, S. Fratini, F. Ingrand, J. Ocon, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. van Winnendael. A Goal-Oriented Autonomous Controller for Space Exploration. In *ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation*, 2011.
6. A. Cesta, A. Finzi, S. Fratini, and A. Orlandini. Enriching apsi with validation capabilities: the keen environment and its use in robotics. In *ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation*, 2011.
7. A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Flexible Timeline-Based Plan Verification. In *KI 2009*, volume 5803 of *LNAI*, 2009.
8. A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Analyzing Flexible Timeline Plan. In *ECAI 2010. Proceedings of the 19th European Conference on Artificial Intelligence*, volume 215. IOS Press, 2010.

9. A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Flexible plan verification: Feasibility results. *Fundamenta Informaticae*, 107:111–137, 2011.

10. A. Cesta, A. Finzi, and A. Orlandini. Using Validation and Verification Techniques for Robust Plan Execution. In *i-SAIRAS-12. Proc. of the 11^{th} Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*, 2012.

11. A. Cesta and S. Fratini. The Timeline Representation Framework as a Planning and Scheduling Software Development Environment. In *PlanSIG-08. Proc. of the 27^{th} Workshop of the UK Planning and Scheduling Special Interest Group, Edinburgh, UK, December 11-12*, 2008.

12. S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. *Integrated Planning and Execution for Autonomous Spacecraft*, volume 1, pages 263–271. IEEE Aerospace, 1999.

13. J. Frank and A. Jonsson. Constraint Based Attribute and Interval Planning. *Journal of Constraints*, 8(4):339–364, 2003.

14. E. Gat. On Three-Layer Architectures. In *Artificial Intelligence and Mobile Robots*. MIT Press, 1997.

15. R. P. Goldman, D. J. Musliner, and M. J. Pelican. Exploiting implicit representations in timed automaton verification for controller synthesis. In *HSCC-02. Proc. of the Fifth Int. Workshop on Hybrid Systems: Computation and Control*, 2002.

16. A. Jonsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in Interplanetary Space: Theory and Practice. In *AIPS-00. Proc. of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling*, pages 177–186, 2000.

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

18. S. Lemai and F. Ingrand. Interleaving Temporal Planning and Execution in Robotics Domains. In *AAAI-04*, pages 617–622, 2004.

19. N. Muscettola, G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt. Idea: Planning at the core of autonomous reactive agents. In *Proc. of NASA Workshop on Planning and Scheduling for Space*, 2002.

20. K. L. Myers. Cpef: A continuous planning and execution framework. *AI Magazine*, 20(4):63–69, 1999.

21. I. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. Shu, and D. Apfelbaum. Claraty: Challenges and steps toward reusable robotic software. *International Journal of Advanced Robotic Systems*, 2008.

22. A. Orlandini, A. Finzi, A. Cesta, and S. Fratini. Tga-based controllers for flexible plan execution. In *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI.*, volume 7006 of *Lecture Notes in Computer Science*, pages 233–245. Springer, 2011.

23. F. Py, K. Rajan, and C. McGann. A systematic agent framework for situated autonomous systems. In *AAMAS*, pages 583–590, 2010.

24. R. Simmons and D. Apfelbaum. A task description language for robot control. In *in Proceedings of the Conference on Intelligent Robots and Systems (IROS*, 1998.

25. B. C. Williams, M. D. Ingham, S. H. Chung, and P. H. Elliot. Model-based programming of intelligent embedded systems and robotic space explorers. *Proc. of the IEEE*, 91(1), jan 2003.