A GOAL-ORIENTED AUTONOMOUS CONTROLLER FOR SPACE EXPLORATION

A. Ceballos[1], S. Bensalem[2], A. Cesta[3], L. de Silva[4], S. Fratini[3], F. Ingrand[4], J. Ocón[1], A. Orlandini[3], F. Py[5], K. Rajan[5], R. Rasconi[3], and M. van Winnendael[6]

[1]*GMV, Isaac Newton 11, P.T.M. Tres Cantos, E-28760 Madrid, Spain, {aceballos,jocon}@gmv.com*

[2]*VERIMAG Centre Équation, 2 avenue de Vignate, 38610 GIÈRES, France, saddek.bensalem@imag.fr*
[3]*ISTC-CNR, Via San Martino della Battaglia 4, I-00185 Rome, Italy,*
{amedeo.cesta,simone.fratini,andrea.orlandini,riccardo.rasconi}@istc.cnr.it
[4]*LAAS-CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 4, France, {felix,ldesilva}@laas.fr*
[5]*Monterey Bay Aquarium Research Institute, 7700 Sandholdt Road, Moss Landing, CA 95039, U.S.A.,*
{fpy,kanna.rajan}@mbari.org
[6]*European Space Research & Technology Centre, Keplerlaan 1, Postbus 299, 2200 AG Noordwijk (The Netherlands),*
Michel.van.Winnendael@esa.int

ABSTRACT

The Goal-Oriented Autonomous Controller (**GOAC**) is the envisaged result of a multi-institutional effort within the on-going Autonomous Controller R&D activity funded by ESA ESTEC. The objective of this effort is to design, build and test a viable on-board controller to demonstrate key concepts in fully autonomous operations for ESA missions. This three-layer architecture is an integrative effort to bring together four mature technologies: for a functional layer, a verification and validation system, a planning engine and a controller framework for planning and execution which uses the *sense-plan-act* paradigm for goal oriented autonomy. **GOAC** as a result will generate plans *in situ*, deterministically dispatch activities for execution, and recover from off-nominal conditions.

Key words: Goal-oriented autonomy, timeline-based planning, interleaving planning and execution, correct by construction modules.

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. Round-trip light time delays and lack of models of remote planetary environments exacerbate the control problem for reliably conducting scientific and engineering operations. 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 inappropriate and even hazardous to the system. Further, most modern spacecraft have on-board control loops that can be reliably closed *in situ* using concepts well established from the field of autonomous science.

To build such controllers however requires dealing with planning and execution time uncertainty and the capability to reason over metric time and resources; *deliberation* is intrinsic, but it also requires immediate *reactive* response to evolving conditions that a robot has to face. In order to deal with these challenges, the lowest level of **GOAC** is a functional layer that is tightly integrated with a decisional layer at the highest level, both of which have a rich history with deployments in complex, real-world environments. The system's higher levels of abstraction deal with long-term mission plans that are *deliberative* whereas lower levels of abstraction are increasingly *reactive*. The functional layer is purely reactive with fast reaction times necessary for failure recovery and command dispatching. Additionally a verification and validation system ensures *correctness-by-construction* of the functional layer, with respect to properties such as deadlock-freedom.

The resulting system will be a state-of-the-art, robust, and verifiable autonomous controller that will provide adjustable levels of autonomy. Moreover it will allow personnel to focus on *what* they want the robotic platform to do instead of laboriously working on *how* to satisfy science and engineering goals.

2. HISTORICAL PERSPECTIVE

For more than two decades, controllers of robotic systems have been mainly using three-layer architectures. Each layer, corresponding to the functional layer, the executive and the deliberative layer, has a different structure, responsibility and interface. The LAAS architecture [10] is an example of such a layered architecture.

The Remote Agent (RA) [14, 17] was the first AI-based closed-loop autonomous control system to take control of a spacecraft. The RA consisted of three different components: a planner-scheduler, an executive, and a fault detection, isolation and recovery subsystem. RA was revolutionary in demonstrating *goal commanding* in space, but the differences in models, syntax and semantics between components led to some design and implementation issues. The Intelligent Distributed Execution Architecture (IDEA) [13] tackled this shortcoming by using a unified planning and execution framework. IDEA was designed with interleaved planning and execution with a collection of controllers within a common framework. However IDEA, did not enforce a systematic framework for formally governing these interactions. The Teleo-Reactive Executive (T-REX) [16, 18] was therefore designed to overcome these restrictions using a collection of controllers called *reactors*. The novelty of T-REX is that it enforces a systematic framework for formally managing the interactions between these reactors, thus responding to the underlying view that both the synchronization of state and dispatching of plan primitives are critical to ensure a correct behavior of the agent and for making the approach efficient and scalable in practice. T-REX is the coordination component of the decisional part of the **GOAC** architecture.

Automated planning was first used on-board a spacecraft in the RA [11], and it was used in a mixed-initiative system for activity planning such as MAPGEN [4] in the Mars Exploration Rover mission, the first AI based system to control a spacecraft on the surface of another planet. MAPGEN uses EUROPA$_2$ [8], a constraint-based planner. EUROPA$_2$ uses a domain model, together with initial conditions and desired goals, to build a temporal network with relations that must be true at start time. It propagates the relations forward and applies goal constraints to select a set of conditions that should be true in the future, some of these corresponding to actions to be undertaken. If a goal cannot be achieved, the planner can backtrack and search for alternatives, and it can also discard unachievable goals. The deliberative reactors of T-REX use EUROPA$_2$ as an embedded planner.

The most advanced planning framework developed in ESA to date is the Advanced Planning and Scheduling Initiative (APSI) [6] system, developed by ISTC-CNR. Built on the lessons learnt from MEXAR2 [5], APSI is a framework to develop mission planning systems based on constraint-based temporal planning techniques and provides the core components and capabilities needed for modelling and solving a planning problem. APSI was selected as the framework for building the on-board planner of **GOAC**. This requires embedding an APSI-based planner into a new T-REX deliberative reactor.

The functional layer of the LAAS architecture is based on the Generator of Modules (G^{en}oM) [7], a mature framework for the development of modules that has been in use in robots for more than a decade. The advantages of G^{en}oM are that it provides a framework for the development of the *functional layer* in a modular way, as well as a set of standard primitives for task communication and management. Moreover, G^{en}oM offers a clear separation between user code (implementing the algorithms) and generic code (functions and automata ensuring the proper execution of the user code).

The complexity of systems currently being built creates many difficulties for system design. These difficulties can be traced in large part to our inability to predict myriad outcomes of robotic action and environmental response. A fundamental idea in systems engineering is that complex systems are built by assembling components, i.e., simpler sub-systems. Designers attempt to solve new problems by reusing, extending and improving past solutions proven to be efficient and robust, which favors component reuse. However, system-level integration is challenging in such circumstance: components are usually highly heterogeneous, have different characteristics, and are often developed using different technologies. Other difficulties stem from current design approaches, which are often based on expertise and experience of design teams. The space environment for which **GOAC** is conceived requires a high reliability and safety of the system as a whole. Therefore, a central problem is the composition of heterogeneous components in a way that ensures their correct inter-operation.

For that purpose, we use the Behavior, Interaction, Priority (BIP) framework [1], developed at VERIMAG, which provides a model-based and component-based methodology to build heterogeneous systems.

Therefore, the functional layer of **GOAC** is an evolution, relying on the formal tool BIP, of the pure G^{en}oM architecture. A good organization of modules (external API) and module components (internal API), which are both provided by G^{en}oM, are a first step in the direction of a *component-based* approach for the development of correct and dependable systems.

3. COMPONENTS OF THE SOLUTION

3.1. Interleaving planning and execution: T-REX

In **GOAC**, planning and execution are inter-twined. T-REX [16, 18] is a *goal-oriented* system, with embedded automated *planning and execution*, encapsulating the long-standing notion of a *sense-deliberate-act* cycle, where sensing, planning and execution are *interleaved*. In order to make embedded planning scalable, the system enables the scope of deliberation to be partitioned functionally and temporally. While T-REX was built for a specific underwater robotics application, the principles behind its design are applicable in any domain where deliberation and execution are intertwined in embedded robotic applications. This is the case, for instance, in robotic systems designed for space exploration missions.

T-REX has strong execution semantics and separates synchronization from deliberation. It further allows synchro-

nization to be interleaved with deliberation permitting reaction times to extend to multiple time steps if necessary. Moreover, the partitioning structure provided automatically applies rules for synchronization and dispatch to coordinate among control loops and resolve conflicts. Safe and effective adaptation requires a balanced consideration of mission objectives, environmental conditions and available resources.

A T-REX *agent* comprises a set of coordinated concurrent control loops. Each control loop is embodied in a *reactor* that encapsulates all details of how to accomplish its control objectives. There is a well-defined messaging protocol for exchanging facts and goals between reactors: *observations* of the current state either from the environment or from within the platform, and *goals* to be accomplished. Reactors are differentiated based on whether they need to deliberate in abstraction (at the highest level) or be responsive to the inputs from the lower levels closer to the hardware. The former therefore will have larger *look-ahead* windows in which to deliberate, while the latter usually do not; and the former will have larger *latencies* within which to return a partial-plan for dispatching to other reactors while the latter will not. Such gradation allows the entire system to be both *deliberative* as well as *reactive* over its temporal scope.

3.2. Timeline-based planning: APSI

Planning in **GOAC** relies on a specialized planner based upon the APSI [6] technology. In particular, being APSI a software development framework for planning and scheduling systems, we have built a planner within such software environment specifically to serve the needs of **GOAC**. The planner, generically referred to as GOAC-APSI-Planner, has functionality very close to the Open Multi-component Planner and Scheduler (OMPS) [9] and in particular uses a refinement of its Domain Definition Language (DDL.3) to describe the relevant constraints of a planning domain.

The GOAC-APSI-Planner uses a native timeline-based representation that is shared with T-REX in order to support a natural integration planner/executor in the decisional layer of **GOAC**. The natural modeling reference for all these planning systems is a *timeline-based representation*, which constitutes a *domain model* as a set of *state variables*. A *timeline* is an evolution of a particular state variable over time. The domain modeling language allows to specify the allowed state transitions as well as the causal and temporal relationships between state variables in order to describe the physics of the specific device we are controlling

It is worth reminding that in tineline-based planning there is no explicit notion of *action* as used in other planning systems. A *planning problem* is presented as a given future desired state of the system, and the outcome of the planning process is the series of state transitions that are required in order to achieve the desired temporally qualified conditions specified as a planning goal.

Similar to APSI and OMPS, the GOAC-APSI-Planner integrates the concept of component as a generalization of state variable, can be configured to model resources, and uses the concept of decision network as a constraint network of decisions on different components, which the planner leverages to propagate new decisions and constraints that are synthesized during planning—the concept of *decision network* can be used alternatively with *current plan* with the same meaning.

3.3. A framework for the functional layer: G$^{en}_{o}$M and BIP

G$^{en}_{o}$M [7] is a development framework specifically intended for the definition and implementation of modules that encapsulate algorithms embedded in target machines such as robotic systems. A *module* is a standardized software entity that is able to offer *services* which are implemented by a set of *algorithms*. Users can start or stop the execution of these services, pass arguments to the algorithms and read the data produced. G$^{en}_{o}$M provides a standard interface to interact with the services and data provided by modules.

Each G$^{en}_{o}$M module of the functional layer is responsible for a particular functionality of the robot. For example, the basic sensors and effectors are managed by their own modules (e.g., one module for the camera pair and one module for the laser range finder). More complex functionalities are encapsulated in higher level modules (e.g., a module doing stereo correlation will use the image taken by the camera module). The most complex functions (such as navigation) can be obtained by having modules "work" together

The BIP framework [1] provides a methodology for building real-time systems consisting of heterogeneous components. BIP is used in order to reduce *a posteriori* validation as much as possible, by putting the focus on the following challenging problems:

- incremental *composition* of heterogeneous components;
- ensuring *correctness-by-construction* for essential system properties such as mutual exclusion and deadlock freedom;
- automated support for component integration and generation of "glue code" meeting given requirements.

G$^{en}_{o}$M along with BIP provides a framework for the development of the functional layer of robotic systems, featuring a modular and leveled structure wherein certain (both intra-module and inter-module) constraints can be enforced at run-time [3].

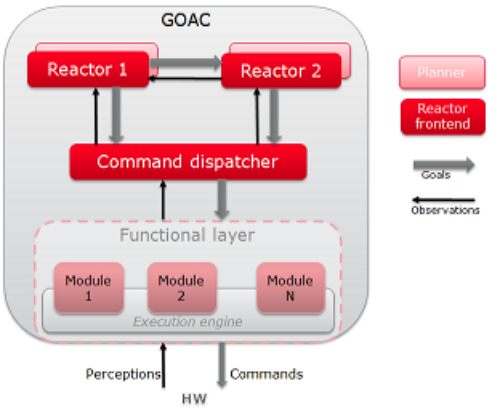

Figure 1. GOAC architecture

3.4. The need for verification and validation: D-Finder

The BIP connectors added to enforce certain properties could cause *deadlocks* in the functional layer, since they amount to adding tighter *constraints* to subsets of components. A deadlock is usually the consequence of over constraining the interaction between components, so the solution is to relax certain constraints.

D-Finder [2] is a tool for compositional deadlock detection and verification in component-based systems described in the BIP framework, with the final objective of proving properties such as *global deadlock freedom*. In the case of a finite-state system, the BIP methodology guarantees that all global deadlocks will be found. In the case of infinite state systems, which is the usual case in practice, the system to be analyzed is an over approximation of the real system. In this case, all actual deadlocks will be found, but some false positives may arise due to using an over approximation of the system. The methodology includes means to identify false deadlocks. We usually follow an incremental verification process: first we verify atomic components and then we verify the compound components resulting from their composition.

4. OVERALL SYSTEM ARCHITECTURE

GOAC is a hybrid architecture consisting of a set of reactors and a functional layer. The architecture is illustrated in Fig. 1. There are several *deliberative reactors* and a *command-dispatcher reactor*. The organization of reactors is not purely hierarchical, but there is a hierarchy of dependencies among reactors. Each deliberative reactor uses a planner based on APSI (background box behind 'Reactor 1' and 'Reactor 2' in Fig. 1). The APSI planner used in **GOAC** comes with the added capabilities of dynamic re-planning and step-wise deliberation. The front-end of a deliberative reactor (red front boxes 'Reactor 1' and 'Reactor 2' in Fig. 1) complies with the messaging protocol of goals and observations between reactors.

GOAC follows a divide-and-conquer approach to complexity, by splitting the deliberation problem into sub-problems, thus making it more scalable and efficient. The number of deliberative reactors to be instantiated in the on-board controller of a given space robotic system is a mission-specific design decision. The set of deliberative reactors represent a partition of the deliberation problem which is both *functional* and *temporal*. Each reactor deliberates over a different part of the domain functionally; for instance, a science reactor and a navigation reactor take into account different aspects of the mission. On the other hand, it is a temporal partition in that the planning horizon is different for each reactor. For instance, whereas a mission-level reactor could consider the whole mission life (e.g., a day-long survey), a navigation reactor would look into the future for a single navigation unit, such as a traverse towards a given target point.

Since the planning problem can be computationally intensive, by splitting it into several sub-problems, we achieve scales of efficiency. If, for instance, a lower-level deliberative reactor can cope with a re-planning need, higher-level reactors will not be informed. In this way, a subset of the planning domain can be used to efficiently satisfy a re-planning need.

Planning in **GOAC** is a model-driven process, in the sense that it relies on a *domain model* representing the world and the robot capabilities. The model comprises a set of state variables, and describes the allowed transitions as well as a set of synchronization rules. Synchronization rules describe causal and temporal relationships between state variables. Domain models are written in a refinement of DDL.3 [9]. Since the planning problem is split into sub-problems, each deliberative reactor uses its own instance of the planner. In turn, each planner uses its own domain model to deliberate over its specific subset of the overall domain. The models of the reactors partially overlap each other, which captures the fact that there are some dependencies across reactors.

Goals follow a top-down flow, from an abstract mission-level reactor, to more specific reactors, down to the command-dispatcher reactor, which translate goals into *requests* to the functional layer and is the interface to the functional layer. Observations flow in a bottom-up direction. At the bottom of the hierarchy, the command-dispatcher reactor synthesizes observations from *replies* received from the functional layer, and forwards these to reactors placed at higher levels of abstraction, which in turn generate more abstract observations. The interaction between both realms follows a request-reply pattern. The command-dispatcher reactor generates requests to be sent to the functional layer from the goals received from other reactors, and converts replies from the functional layer into observations for other reactors.

The functional layer is based on G^{en}oM and BIP. The basic self-contained design unit in G^{en}oM is a module. Each module encapsulates a functionality of the robotic system. The definition of a module includes a well-specified interface consisting of services and exported data, called

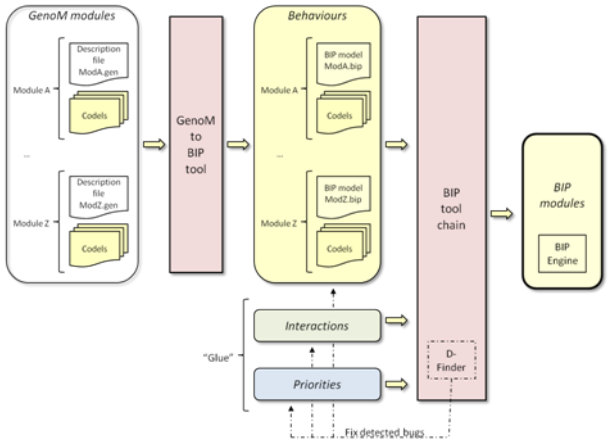

Figure 2. G^{en}oM/BIP

requests and *posters*, respectively. The set of modules comprising the functional layer of a robotic system is mission specific, especially for modules responsible for controlling hardware elements. However, like hardware devices, functional modules are reusable across different robotic platforms.

The definition of a G^{en}oM module describes the computational model it needs, and the execution model of its services, while the *BIP model* of a module provides a formal description of the module's behavior. On top of the modules BIP models, it is possible to specify inter-module and intra-module synchronization and causality constraints. The G^{en}oM/BIP framework favors a development focused on the implementation of the *codels*, the algorithmic coding units, rather than lower-level details, such as multi-thread synchronization or data sharing. A specific tool, the G^{en}oM-to-BIP translator, automatically generates a BIP model from a G^{en}oM module definition. The joint G^{en}oM/BIP development process is depicted in Fig. 2.

In order to access the functional layer modules, the G^{en}oM framework has a native API in the C language, and provides bindings for Tcl and for OpenPRS [10]. In **GOAC**, the command-dispatcher reactor uses an OpenPRS-based interface for the interaction with the functional layer. This interface provides decoupling and a more convenient level of abstraction to the deliberative and executive layers.

The main processing function of the controller indefinitely repeats a sense-deliberate-act control cycle. On every cycle, planners accept goals from higher-level reactors or from an external system such as the ground control center; deliberative reactors post goals to lower-level reactors as a result of its planning activity; observations are synchronized to guarantee a common view of the world at the execution frontier; but most computation effort is reserved for deliberation if needed. In **GOAC**, time is assigned to the different planners as *steps of deliberation* or time slots, according to a reactor precedence based on previous assignments and pending planning work.

Plans in **GOAC** are *flexible*: goals can be flexibly executed because the start time, the end time and the duration are time intervals. This way, plans are more robust in the frame of uncertain environmental conditions, than predefined, rigid sequence of activities.

In addition to goal-based operations, **GOAC** supports all autonomy levels as defined by the ECSS [19], enabling a complex mission configuration relying on mission-level (or lower-level) goals, on-board control procedures (OBCP), event-driven actions, time-tag commands and tele-operation. **GOAC**'s flexible design enables a mission concept wherein autonomy can be dynamically and incrementally delegated from ground to space.

GOAC is an architecture that defines a template that must be instantiated for a specific robotic platform. The instantiation encompasses a number of development tasks of a different nature. It can be outlined in the following steps, which are a bottom-up description of the process:

- definition of modules of the functional layer and implementation of the codels;
- generation of BIP models of the modules;
- addition of constraints between modules;
- verification of essential system properties such as, but not limited to, deadlock freedom;
- implementation of the OpenPRS procedures representing the access to the functional layer and making available an API to the reactors;
- implementation of the command-dispatcher reactor; and
- implementation of the domain models for the deliberative reactors.

5. CASE STUDIES

In the scope of the Autonomous Controller activity, two case studies have been selected in order to demonstrate the capabilities of **GOAC**. The first case study is based on a terrestrial rover testbed, and the second is based on a rover simulator. The selection is the result of a trade-off between sharing commonalities and efforts across both studies and showing GOAC in two different scenarios.

5.1. DALA: An exploration rover

The DALA robot is one of the LAAS robotic platforms that can be used for autonomous exploration type of experiments. DALA is a mature iRobot ATRV robot that provides a large number of sensors and effectors. It can use vision based navigation (such as the one used on Spirit and Opportunity), as well as indoor navigation

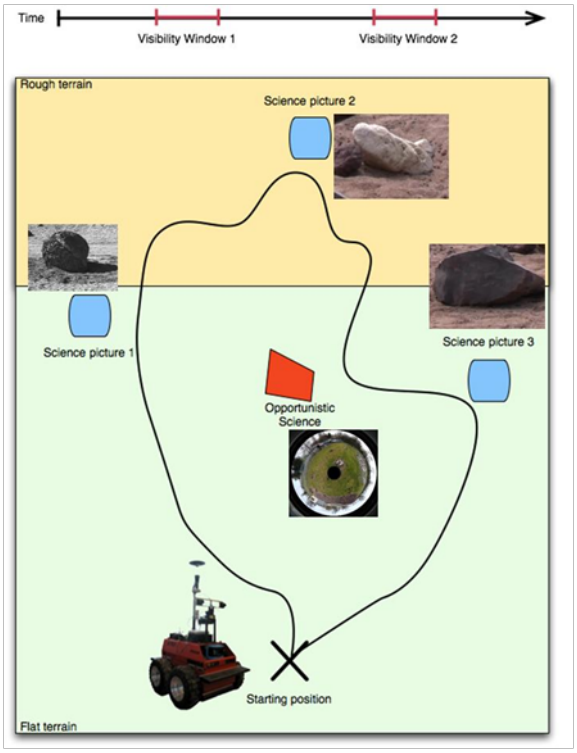

Figure 3. Scenario proposed for the DALA robot

based on a Sick laser range finder. DALA is a mature platform that has been successfully used for several years.

The purpose of the DALA scenario is basically to behave as close to a Mars or Lunar rover as possible. To this effect, the rover has a number of concurrent tasks to achieve:

1. Navigate safely in an a priori unknown environment.
2. Take high resolution pictures of an operator-given list of locations (presumably because of some scientific interest).
3. Communicate with some orbiters or a lander during some given visibility windows.
4. Continuously monitor the environment for "opportunistic science," and take appropriate actions when something interesting is detected.
5. Monitor and control the proper heating of the platform and the payload.
6. Monitor and control the proper power usage and energy consumption.

A typical scenario involving these different activities could be as follows. At the beginning of the day, the rover wakes up and warms up. It has a mission to take high-resolution pictures (possibly stereo) of three different locations, which have been chosen by scientists based on previously acquired data. Moreover, there are two communication "orbiter visibility" windows during this day when the rover establishes a connection to transmit recently acquired data, and to receive further instructions. Communication is done while the robot is still to mitigate power usage. Meanwhile, while navigating to the different locations, the rover continuously monitors the environment (with a panoramic camera) for possibly interesting scientific features. When such a feature is detected, the robot takes a picture and pinpoints the location of the feature to later send this information to the ground (for review and further investigations in future missions). Finally, the rover navigates back to its starting point before the end of the day. All of this is performed while the robot monitors its energy level as well as its power usage.

The implementation of the **GOAC** controller for DALA started at the functional layer. Firstly, G^{en}oM modules for all devices were incrementally integrated: starting from basic hardware functions in charge of power and locomotion, then laser-based indoor navigation and PTU and cameras, and finally the more complex stereovision-based outdoor navigation. At a second major stage, the G^{en}oM modules were migrated to BIP, thus enabling the definition of inter-module and intra-module constraints to be incorporated into the controller behavior, as well as the detection of deadlocks by D-Finder.

The **GOAC** framework includes tools to incrementally test components: isolated modules can be tested by individually examining the services they offer, and all integrated modules can be tested by means of a Tcl-based or OpenPRS-based supervisor. At the same time, the deliberative/executive layer has been implemented and integrated with the functional layer. Initially, a single deliberative reactor handled only a simple domain model. OpenPRS procedures were initially used to simulate the functional layer, thus closing the lower loop. After separately testing the functional layer and the deliberative/executive layer, they were put together for a system test running a simplified version of the scenario. A mission-level goal was injected at system startup which led to the execution of a simplified experiment like the one described above.

This scenario gives the opportunity to demonstrate **GOAC** by using goal commanding and by controlling real hardware.

5.2. 3DROV: A rover simulator framework

The 3DROV [12, 15] is a design and simulation tool for planetary exploration rovers, particularly suitable for rover simulation. It includes models of the planetary environment; the mechanical, electrical, and thermal subsystems of the rover; generic on-board controller, and the ground control station. Fig. 4 shows a snapshot of the 3DROV visualization tool. The simulation environment is based on ESA's SIMSAT 4.0. The models are compliant with SMP 2.0.

In the 3DROV scenario, we use **GOAC** to control the

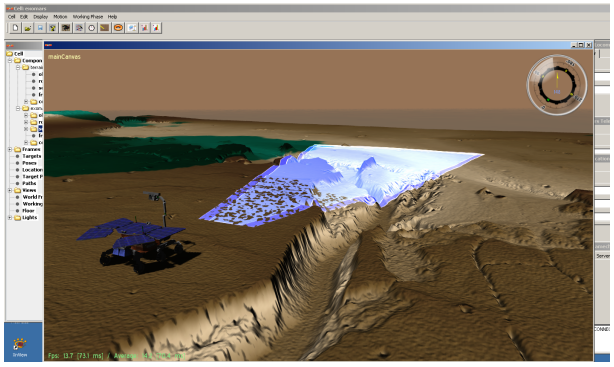

Figure 4. 3DROV visualization tool, showing camera footprints and overlay

rover simulator. The scenario includes a Mars-like terrain with small craters and cliffs. The overlay functionality of the 3DROV is used to represent certain areas of potential geological interest.

The integration of **GOAC** in the 3DROV involves modifying the 3DROV generic controller. In order to do that, the 3DROV instance of the **GOAC** uses a software library that makes available an interface to the algorithms of the generic controller.

The SIMSAT environment is based on GNU/Linux shared libraries for the different pieces to be used in a simulation. In particular, a specific shared library encapsulates the controller. Therefore, the **GOAC** instance of the 3DROV must provide a shared library to be loaded on SIMSAT. In order to do that, a special module of the functional layer called Algo3drov has been implemented. This module complies with the G^{en}oM framework so that in can work together with the rest of the functional layer and the module uses the 3DROV actions library. Two separate components address both functions, and they are inter-connected by means of remote procedure calls (RPC).

The resulting architecture completely relies on the existing algorithms of the generic controller of the 3DROV for low level access to the simulation. Yet, from the point of view of G^{en}oM/BIP users (T-REX and OpenPRS), it has the structure of a G^{en}oM/BIP module. Moreover, the architecture also uses existing G^{en}oM modules for stereo based navigation (those modules are shared between the DALA and 3DROV experiments). As a result, it is possible to specify BIP safety constraints over all these functional modules.

Several scenarios of increasing complexity have been designed for the 3DROV. Successive scenarios are intended to demonstrate the different **GOAC** capabilities, such as re-planning or BIP constraints, dynamic injection of mission goals and adjustable autonomy levels.

6. CONCLUSIONS

GOAC responds to the needs of space missions in which higher degrees of on-board autonomy are required.

The model-driven methodology underlying **GOAC** allows an incremental design and implementation of controllers for space robotic systems, and naturally decouples major system components, which can be implemented by different development teams.

A timeline-based planning approach produces flexible plans that can cope to a large extent with environmental activities without the need of frequent re-planning. Planning is a continuous activity, which, by splitting the planning domain model into sub-models, can be highly efficient. The flexibility of timeline-based plans together with the re-planning capability provides a robust way to face the uncertainty inherent in space exploration missions.

A formal description of the modules comprising the functional layer enables the detection and verification of essential system properties such as deadlocks, which eventually leads to a *correct-by-construction functional layer*.

By combining different areas in Artificial Intelligence, **GOAC** fosters an operational concept in which operators can focus on the *domain level* rather than lower-level aspects of the mission. In addition to potentially reducing the costs of the mission as a consequence of *goal-oriented operations*, a high-level of autonomy also inproves the *performance* of the robotic platform in two ways:

- scientific performance: high-level guidelines for nominal as well as opportunistic scientific exploration allows being more efficient in doing science than relying on pre-calculated plans, potentially resulting in a higher mission return; and
- robustness of the robotic system: when a robotic system reaches off-nominal situations it can be safer and more efficient to autonomously react to the environmental conditions and autonomously re-plan rather than wait for ground instructions or rely on fixed pre-programmed alternative plans, potentially increasing the probability of successfully meeting the mission's objectives.

REFERENCES

[1] A. Basu, M. Bozga, J. Sifakis, Modeling Heterogeneous Real-time Components in BIP. In 4th IEEE International Conference on Software Engineering and Formal Methods, Washington DC, USA, 2006, *IEEE Computer Society*.

[2] S. Bensalem, M. Bozga, J. Sifakis, T.-H. Nguyen. D-Finder: A Tool for Compositional Deadlock Detection and Verification. Computer Aided Verification,

21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings. *Springer, Lecture Notes in Computer Science* 5643

[3] S. Bensalem, L. de Silva, M. Gallien, F. Ingrand, R. Yan. "Rock Solid" Software: A Verifiable and Correct-by-Construction Controller for Rover and Spacecraft Functional Levels. *International Symposium on Artificial Intelligence, Robotics and Automation for Space (2010)* Sapporo, Japan.

[4] J. Bresina, A. Jonsson, P. Morris, and K. Rajan. Activity Planning for the Mars Exploration Rovers. In *ICAPS-05. International Conf. on Automated Planning and Scheduling*, Monterey, California, 2005.

[5] A. Cesta, G. Cortellessa, M. Denis, A. Donati, S. Fratini, A. Oddi, N. Policella, E. Rabenau and J. Schulster. MEXAR2: AI Solves Mission Planner Problems. In *IEEE Intelligent Systems*, 22(4):12-19, 2007

[6] A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi. Developing an End-to-End Planning Application from a Timeline Representation Framework. In *IAAI-09. Proceedings of the 21st Innovative Applications of Artificial Intelligence*, 2009

[7] S. Fleury, M. Herrb and R. Chatila. G$^{en}_{o}$M: A Tool for the Specification and the Implementation of Operating Modules in a Distributed Robot Architecture. In *International Conference on Intelligent Robots and Systems*, pages 842-848. Grenoble (France), 1997.

[8] J. Frank and A.K. Jónsson, Constraint-based Attribute and Interval Planning, In *Constraints*, 8, 4, 339-364, 2003.

[9] S. Fratini, F. Pecora, and A. Cesta. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. In *Archives of Control Sciences*, 18(2):231-271, 2008

[10] F. Ingrand, S. Lacroix, S. Lemai-Chenevier and F. Py, Decisional Autonomy of Planetary Rovers, *Journal of Field Robotics*, Volume 24, Issue 7, Pages 559 - 580, July 2007

[11] A.K. Jónsson, P. Morris, N. Muscettola, K. Rajan and B. Smith, Planning in Interplanetary Space: Theory and Practice, *AIPS*, Breckenridge, CO, 2000

[12] K. Kapellos and L. Joudrier, 3DROV: A Planetary Rover Design Tool based on SIMSAT v4. In *ESAW2009*, ESA/ESOC, Darmstadt, Germany, May 2009

[13] N. Muscettola, G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt, IDEA: Planning at the core of autonomous reactive agents" in *Proc IWPSS*, Houston, 2002

[14] N. Muscettola, P. P. Nayak, B. Pell and B. C. Williams, Remote Agent: To Boldly Go Where No AI System Has Gone Before, in *AIJ*, 103, 1998

[15] P. Poulakis, L. Joudrier, S. Wailliez, K. Kapellos : 3DROV : A planetary Rover System Design, Simulation and Verification Tool. *9th International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSairas)*, Hollywood, USA, February 26 - 29, 2008

[16] F. Py, K. Rajan & C. McGann. A Systematic Agent Framework for Situated Autonomous Systems. *9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2010, Toronto, Canada.

[17] K. Rajan, D. Bernard, G. Dorais, E. Gamble, B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, P. Nayak, N. Rouquette, B. Smith, W. Taylor, and Y. Tung, Remote Agent: An Autonomous Control System for the New Millennium. In *PAIS-00. Proc. Prestigious Applications of Intelligent Systems*, ECAI, Berlin, Germany, 2000.

[18] K. Rajan, F. Py, C. McGann, J. Ryan, T. O'Reilly, T. Maughan & B. Roman. Onboard Adaptive Control of AUVs using Automated Planning and Execution. *Intnl. Symposium on Unmanned Untethered Submersible Technology (UUST)* August 2009. Durham, NH

[19] ECSS – Space engineering – Space segment operability, ECSS-E-ST-70-11C, 31/07/2008