TOWARDS GOAL-ORIENTED SPACE ROBOTICS OPERATIONS USING AUTONOMOUS CONTROLLERS

Antonio Ceballos Roa[(1)], Michel van Winnendael [(2)] , Alberto Medina Andrés [(1)], Jorge Ocón Alonso [(1)], Fernando Gandía Abellán [(1)]

[(1)] GMV, Isaac Newton 11, Tres Cantos, 28760 Madrid, Spain, e-mail:{aceballos,amedina,jocon,fgandia}@gmv.com
[(2)] ESA/ESTEC TEC-MMA PO Box 299, 2200AG Noordwijk, Netherlands, e-mail:Michel.van.Winnendael@esa.int

ABSTRACT

The Goal-Oriented Autonomous Controller (GOAC) has been developed in the frame of an ESA-funded R&D activity with the objective to design a generic architecture that can be used in on-board controllers of space robotics system in which autonomy is important. GOAC encompasses planning capabilities, interleaved planning and execution, a modular functional layer, and online validation and verification. It allows an operations concept based on goals, thus enabling operators to tell the system what to do rather than how to do it. This paper describes the results of the project.

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 accurate models of remote planetary environments exacerbate the control problem for reliably conducting scientific and engineering operations. This is not only because fast reaction is often 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.

Three different factors have a strong influence on the degree of autonomy required for a mission:

- communication delays;
- operations team size and costs;
- uncertainty about the operating environment.

2. GOAC APPROACH TO AUTONOMY

One of the main design drivers of GOAC is to enable goal-oriented operations, which corresponds to the highest level of autonomy currently defined in the European ECSS Space Segment Operability Standard [1], called E4. The E4 level features the capability of making decisions in a way closer to the way humans do. Therefore, instead of writing complex programs, space software engineers have to formally describe a network of relationships between mission elements, so that the space robotic system can reliably make decisions based on that network.

In the frame of space missions, the capability to make decisions, i.e. to deliberate, involves the capability of planning, i.e. to make plans. A plan is a set of tasks that must be executed in order to satisfy a mission objective. In the literature on artificial intelligence, planning is a specific type of problem solving. A planner (problem solver) takes a mission objective (problem) as input and plans for it (solves it), based on a domain description, to eventually produce a plan (solution). In GOAC, planning can be seen as the projection into the future of the desired state of the system. A plan is a set of steps that must be followed in order to reach that desired state.

GOAC defines an architecture for the implementation of autonomous on-board controllers. From a general perspective, the desired capabilities of such controllers are the following:

- autonomous planning and scheduling;
- reactiveness to dynamic external and internal conditions in the present;
- efficiency in simultaneous deliberation and execution;
- performance driven by goals assigned from the outside (ultimately by humans) and by knowledge of the context;
- robustness, i.e. able to deal with off-nominal conditions;
- adjustability of the autonomy level;
- open architecture;
- deterministic behaviour.

To build such controllers 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. GOAC has been designed to be applicable to a wide range of domains.

3. GOAC ARCHITECTURE

The key to the E4 autonomy level in GOAC is the ability of the on-board controller to make decisions by itself, which involves the capability of planning. Traditional approach in classical three-layered architectures conceives planning as a standalone activity, essentially decoupled from plan execution. On the other hand, planning typically involves addressing NP-complete problems, which, despite all efforts to solve them efficiently, tend to be very time consuming. GOAC tackles both shortcomings.

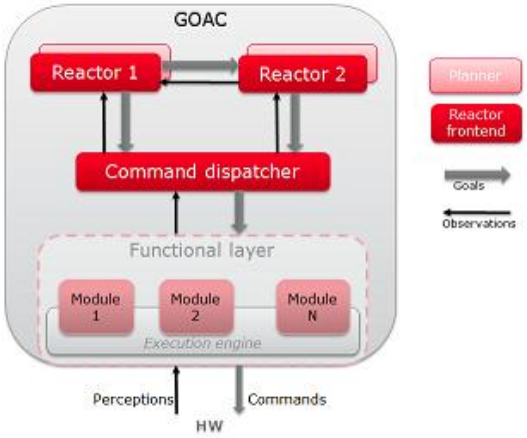

Figure 1. GOAC architecture (example with 2 reactors)

GOAC is a hybrid architecture consisting of a set of so-called reactors and a functional layer. The architecture is illustrated in Fig.1. Each reactor (or teleo-reactor, according to T-REX [2][3] terminology) is a sense-plan-act control loop working at a different level of abstraction and making decisions over a possibly different functional scope of the system. 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. 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.

The intelligence of the controller is in the planner, the problem solver, but instead of a single planner there can be several planners in GOAC. In that case, each planner is embedded into a different reactor. At the bottom of the deliberative-executive layer, there is a command-dispatcher reactor, which is purely reactive, i.e. non-deliberative, it does not produce any plans. The controller can be seen as a hierarchical network of reactors, where the output of each reactor (a plan) is the input (a set of goals) for another reactor.

Therefore, GOAC follows a divide-and-conquer approach to complexity, by splitting the deliberation problem into sub-problems. 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.

Deliberation in GOAC relies on APSI [4][5], which is a framework for the implementation of timeline-based planners. With this technology, plans are flexible, i.e. the time boundaries (start, duration and end) of planned tasks are not fixed. This way plans are more robust in the frame of uncertain environmental conditions than predefined, rigid sequence of activities.

The functional layer is based on GenoM [6][7] and BIP [8][9]. GenoM 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. GenoM provides a standard interface to interact with the services and data provided by modules. Each GenoM module of the functional layer is responsible for a particular functionality of the robot. Complex functions, such as navigation, can be obtained by having modules "work" together.

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 BIP framework 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.

GenoM along with BIP provide a framework for the development of the functional layer of robotic systems, featuring a modular and levelled structure wherein certain intra-module as well as inter-module constraints can be enforced at run-time.

Further details on the design of GOAC are provided in [12].

4. DESIGN OF THE CASE STUDIES

Two implementations of the generic architecture were made as case studies, using laboratory equipment.

4.1. DALA Case Study

The implementation of the GOAC controller on the physical rover DALA [13] of project partner LAAS (Toulouse, France) started at the functional layer. First,

GenoM modules for all devices were incrementally integrated: starting from basic hardware functions in charge of power and locomotion, then laser-based indoor navigation for flat terrain, PTU and cameras, and finally the more complex stereovision-based outdoor navigation for rough terrain. At a second major stage, the GenoM modules were migrated to BIP, thus enabling the definition of inter-module and intra-module constraints to be incorporated into the controller behaviour, as well as the detection of deadlocks by D-Finder [10].

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. Open-PRS [11] 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 enabled execution of a simplified experiment. This provided the opportunity to demonstrate GOAC by using goal commanding and by controlling real hardware. Fig. 2 shows the environment of the scenario.

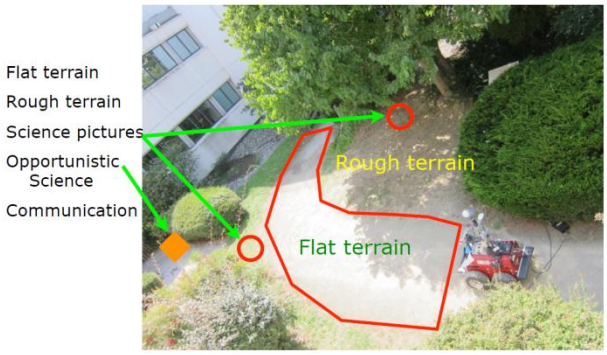

Figure 2. DALA Demonstration Environment

The architecture of this controller implementation is shown in Fig. 3. The BIP engine runs the entire functional layer. Some intra- and inter-module constraints have been included to be enforced by the BIP engine, for instance: only one navigation mode can be active at any given time; the PTU can move only if the robot base is still.

Some runs were conducted explicitly showing the BIP engine capturing forbidden requests, such as a request to use the laser while doing stereovision navigation or a request to move the PTU while the rover is moving. In all cases, the BIP engine discarded the forbidden requests.

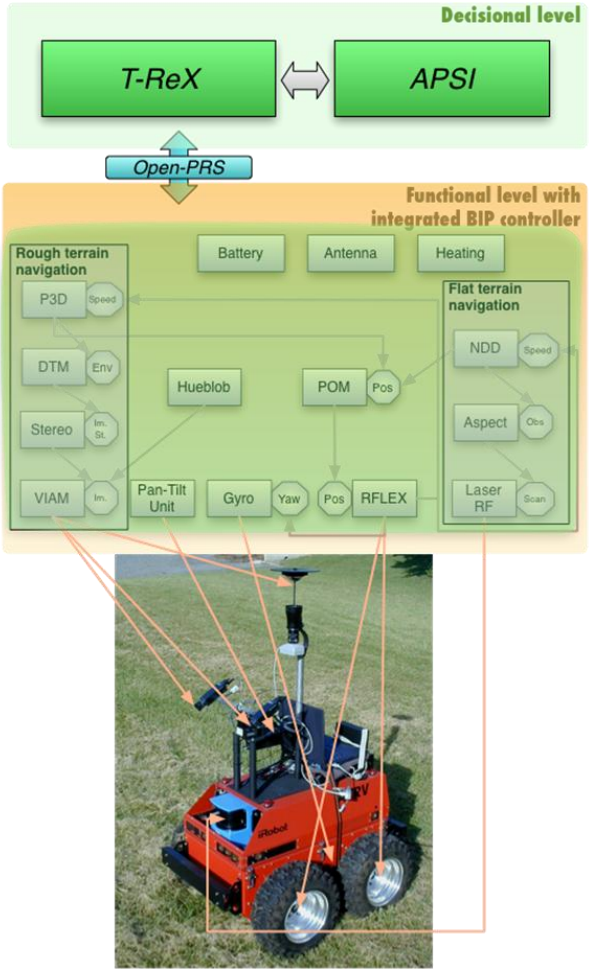

Figure 3. DALA Controller Architecture

4.2. 3DROV Case Study

The integration of GOAC in the 3DROV rover simulator of ESA [14][15] required an adaptation of the generic controller of 3DROV. 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 3DROV simulator uses SIMSAT [16], while 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 GenoM framework so that it 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 GenoM/BIP users (T-REX and OpenPRS), it has the structure of a GenoM/BIP module. It is depicted on Fig. 5.

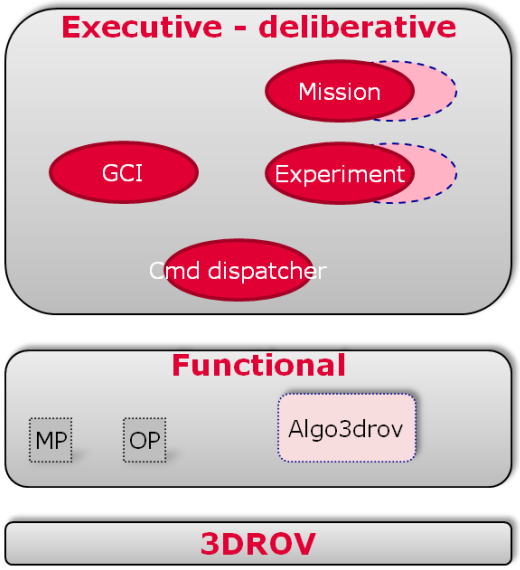

Figure 5. Architecture of the GOAC instance for Scenario J of the 3DROV case study

While the DALA case study is focused on demonstrating E4 (goal-oriented operations), the 3DROV case study also includes demonstrations for the lower autonomy levels. A supporting tool has been used in order to illustrate how a classical spacecraft monitoring and control system could be used to monitor and control a GOAC controller. See Fig. 6 for the overall architecture.

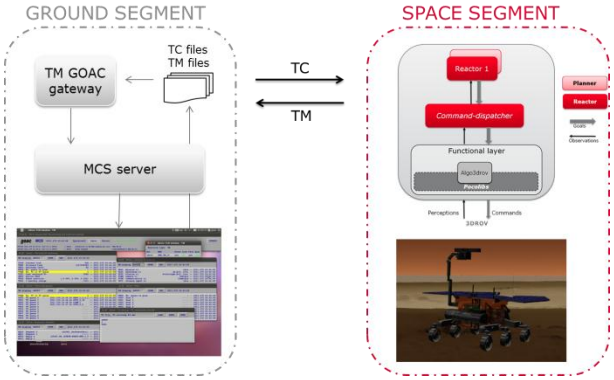

Figure 6. Overall architecture of the 3DROV case study including the ground segment

5. CHARACTERIZATION OF THE CASE STUDIES

A characterization was done of the performances of both implementations described in the previous section.

5.1. DALA Case Study

The instance of GOAC that is used for controlling the DALA rover has been tested on a system with the following characteristics:

The operating system is Fedora 12. The main characteristics of the underlying hardware processor are these:

- Processor model: two Intel Pentium 4, 3.06GHz CPUs, with 512 KB cache each.
- Total memory: 1 GB.
- Linux version: 2.6.32.19-163.fc12.i686 (gcc version 4.4.2).

The runtime performance of BIP and GenoM is shown in Table 1. The results shown are the CPU times taken by each module, obtained via the TIME+ column of the top Unix command. The results are an average over five runs. The experiment conducted was simpler than the one described earlier in that the rover only explored a single location, rather than multiple locations, before returning back to the initial location.

Module	GenoM execution time	BIP execution time
LaserRF	120	1947
Aspect	192	2362
NDD	43	2009
RFLEX	168	10763
Antenna	56	1102
Battery	69	1219
Heating	92	1029
PTU	126	2394
Hueblob	690	1850
VIAM	1046	3225

Table 1. GenoM and BIP execution times for DALA

From the results, we can see that the total time taken by the BIP modules is 27927 time units, whereas the total time taken by the GenoM modules is 2731 time units. Hence, BIP took approximately ten times more CPU time than GenoM.

Likewise, in terms of the usage of the CPUs immediately before the end of the experiments, the GenoM experiment used approximately ten times less than the BIP experiment, with an average usage of 6.3% for the former and 52% for the latter. This additional overhead with BIP comes from the decision making by the BIP engine. In particular, the BIP engine actively computes all the feasible interactions at every cycle, where the time taken for this computation is proportional to the number of interactions in the BIP modules. The number of interactions, however, is not the only factor that affects the speed of modules. Currently, the BIP engine runs continuously without halting, even when there are no interactions to fire. As a result, it uses all the CPU it can get; simply forcing the BIP engine's main loop to run at 25 Hz divides the CPU load by two. Another possible improvement is to be smarter about the guards which are evaluated at each loop, e.g., to only check those in which the associated variable values have changed. Even better, the use of the real-time (RT) BIP engine would dramatically

improve these figures. Some test on the antenna modules using the BIP RT engine show at least one order of magnitude improvement on the overall load. However, as of the time of the experiment, the BIP RT engine was not available with multi-threading, which is required by GenoM modules.

Despite the CPU overhead incurred by using BIP, the time taken to run the complete experiment was approximately 4.5 minutes on average in both GenoM and BIP, with BIP taking about 4 seconds longer on average. Therefore, the overhead in using BIP was negligible when taking into account the total time for executing the mission. This is most likely because in the BIP experiment the CPUs are used at a much higher capacity (52% usage) than in the GenoM experiment (6.3% usage), which would compensate for the extra CPU time needed by the BIP modules. The additional CPU time required by BIP was also perhaps mitigated by the time taken up in executing actions in the real world, such as moving the robot, communicating, and moving the PTU. In our experiments, simply moving DALA from (x, y) coordinates (0, 0) to (4, 0) takes approximately 30 seconds, moving the PTU toward the left or right front wheels of the rover takes approximately 5 seconds, and transmitting a single picture (to the "orbiter") also takes approximately 5 seconds.

As far as performance is concerned, there are a number of parameters that should be taken into account when developing a functional level using this framework. We will discuss the most important ones below.

(1) Since codels of GenoM modules are meant to be the smallest entity of a module, and in particular being entities that cannot be interrupted, codels should be written with appropriate care. Algorithms should be split into codels in a way that each of them handles a specific functionality of the original algorithm without using up too much CPU time.

(2) Likewise, to have more fine grained control over posters, updating the posters of a module could be handled manually by the user from within codels, rather than automatically by GenoM. Manual control of posters allows for better interleaving between multiple entities that need access to a poster at the same time. Efficiency improvements resulting from such manual management of posters in GenoM modules will be reflected in the corresponding BIP modules generated.

(3) When adding BIP modules to a top-level (root) compound BIP component for the purpose of adding inter-module high-level constraints between them, care should be taken to avoid also including BIP modules for which no inter-module constraints will be needed. Although, in theory, any number of BIP modules could be included into the top-level BIP component, even with no high-level constraints defined between them, this may result in reduced performance since all modules will then be handled by a single BIP engine, rather than by multiple separate BIP engines.

(4) The size of a generated BIP model/component (from a GenoM module) can be reduced by being specific about which of the requests belonging to the GenoM module should be included in the corresponding BIP model. This is done by specifying the necessary request names as a list when invoking the "genom-to-bip.rb" command from the command line. This improvement should also, in theory, increase the efficiency of resulting BIP modules since the BIP engine will not need to reason about the unnecessary requests.

(5) The designer should keep in mind that there is a limitation in the "genom-to-bip" tool in which a GenoM request can have at most one activity associated with it in the corresponding BIP model (as opposed to any number of activities in the GenoM module). This limitation entails that whenever the same request is sent multiple times before the completion of a previous "instance" of the request, the new request will abort its previous instance. In other words, only one instance of a request type can be active at any given point during the execution of the associated BIP model. While this is indeed limiting, it can be overcome to a certain extent by defining additional execution service types in the GenoM module (which yields additional Execution Service components in the corresponding BIP module) to cater for the additional activity instances that may be needed at run- time. For example, when simulating battery usage, there is a request called StartUsingBattery - which takes the device, e.g., camera, as an argument - to register the fact that the device in question has started to use the battery, and another request called EndUsingBattery to register the fact that the device has stopped using the battery. Since this design may result in multiple GenoM activities at runtime for the execution service associated with the StartUsingBattery request (specifically, if multiple devices need to concurrently use the battery), we instead have multiple requests, such as CameraStartUsingBattery, one for each device that needs to use the battery.

5.2. 3DROV Case Study

The instance of GOAC that is used for controlling the 3DROV has been tested on a system with the following characteristics:

The operating system is Ubuntu 11.04 (Linux version 2.6.38-8-generic, gcc version 4.5.2), which is running inside a VirtualBox virtual machine, in turn running inside an Ubuntu 9.10.

The main characteristics of the virtual machine are these:

- Base memory: 768 MB
- Virtual disk: 32 GB

The main characteristics of the underlying hardware processor are these:

- Processor model: Intel(R) Core(TM)2 Duo CPU E8400 at 3.0 GHz
- Total memory: 2 GB

- Linux version: 2.6.31-22-generic (gcc version 4.4.1)

A simple scenario was implemented using 3DROV. In this scenario, a mission-level goal called 'Survey J' is requested at startup, which consists of two consecutive experiments: 'Experiment B' and 'Experiment C'.
When the GOAC instance starts, the initial goal defined in the TREX configuration file is sent to the planner. Few seconds later, the plan is ready for execution. At some point while doing Experiment B, the state of the battery changes from high into medium. This triggers replanning in the Experiment (Nav) reactor. The latter fails to produce a new plan for Experiment B, and an Experiment B failure is reported as an observation to the Mission reactor, which successfully replans Survey J.
The CPU and memory usage have been monitored during a run of the software, both the total usage and the usage per process.
The following was observed:

- The base use of CPU when GOAC is not running is 3%.
- The average use of CPU by GOAC is 35% (38% - 3%).
- The use of CPU is relatively stable. The peaks correspond to the periods of time when the system is planning.
- The base use of memory when GOAC is not running is 77%, i.e. 590 MB.
- The average use of memory by GOAC is 14% (91% - 77%), i.e. 108 MB.
- The APSI deliberative reactor and embedded planner use more memory than the other processes. This is mainly due to the Java Virtual Machine.
- There are some peaks of CPU usage by the APSI deliberative reactor and embedded planner, which correspond to the periods when the planner is actively planning, for the initial goal or replanning. After that, the use of CPU by the planner is very low.
- The use of CPU by Algo3drov, TREX and the message passer is negligible.
- The use of memory by Algo3drov and the message passer is negligible.

The look-ahead (temporal scope of deliberation) and the latency (the time needed to generate a plan) of deliberative reactors play an important role in the performance of the on-board controller. Synchronization rules in the DDL.3 domain model have a significant impact on the time it takes the planner to produce a plan, in particular on the size of the search space. If a rule is too restrictive, it may be impossible to find a solution in a particular case. On the other extreme, if a rule is too loose, the space search may grow too large and finding a solution would take longer than the allowed latency.
In the case of the 3DROV, the functional layer strongly relies on the existing generic controller of the 3DROV. Most of the codels of the Algo3drov module are hooks to the corresponding actions of the generic controller. Therefore the performance is not representative of a fully GenoM/BIP functional layer such as that of the DALA Case study.
No latencies have been identified in the interfaces between the different system components.

6. CONCLUSIONS AND FUTURE WORK

6.1. Advantages

Interleaving planning and execution
In classical three-layer architectures, planning and execution are separate tasks within the controller. First, this means that it takes the controller some time to produce a plan. Once a plan is produced, it is dispatched for execution. If, for some reason, the plan fails, the executive layer must stop executing and the planning layer must produce a new plan. This approach can be inefficient. In GOAC, planning and execution are tightly coupled, allowing a more efficient way of replanning when needed.

Timeline-based planning and domain modeling
Timeline-based planners provide a much better handling of uncertainty, because plans are temporally flexible. They reduce the size of the code and its complexity, and they embed this complexity in the planning model, which in our case was described using the DDL.3 language.
The DDL.3 language is used in GOAC for the definition of the possible behaviours of the robotic system under control in its operational environment (domain theory). By relying on domain models, the planner remains a generic software component of the controller. In addition to defining a domain theory, it is possible to embed the definition of a mission in the models. Since models are interpreted by the planner when the controller software starts, it is easy to change the possible behaviours of the robotic system by just uploading a new model.

GenoM
GenoM is a mature framework for the implementation of the modules of the functional layer. It enforces a particular way of doing things, and allows a developer to focus on the algorithms, because the framework handles all aspects related to the communication between modules. In addition to the implementation of the algorithms, the developer just needs to specify the module definition, including the module interface: offered services and exported data.

Correct-by-construction techniques
Real-time verification (BIP) guarantees that the functional layer will not commit an action that could harm the robot or personnel. Many classical formal techniques that are used to verify that a system is correct rely on model checking, which has to face the state

explosion problem. Hence those techniques are not scalable. Describing the functional layer by means of the BIP formal language, it is possible to perform offline analysis of the modules. On the other hand, it is possible to express constraints that are guaranteed at run-time.

Reuse of legacy software

The advantage of using existing software assets is that we have based the solution on solid developments, each of them solving a particular problem (TREX: interleaved planning and execution, APSI: timeline-based planning, GenoM: solid, component-based functional layer, BIP real-time verification).

Project life cycle

For the development the spiral/iterative life cycle seems more appropriate than waterfall in research/innovative projects. We have followed an iterative life cycle for the implementation of the case studies. This was possible because the technical solution had been to a large extent designed upfront. The implementation of both case studies have been largely overlapped in terms of schedule, the DALA case study being more realistic, since it is based on a real robotic asset, and not a simulated one. The synergy between both cases has been clear: many approaches or specific pieces of software have been reused in the 3DROV case study from the DALA case study, and the other way around: some of the 3DROV solutions case study have been reused in DALA. The 3DROV case study encompassed its own challenges, mainly the need for adapting the existing 3DROV Generic Controller to GenoM. It showed that autonomy levels E1 to E4 can be handled. Finally, lessons learned in the implementation of both case studies have provided useful feedback on the generic architecture.

6.2. Conclusions

The Autonomous Controller activity has demonstrated that a Goal-Oriented Autonomous Controller for space robotics systems is technically feasible.

In terms of outcome, the activity has yielded three main products:

- an architecture for on-board controllers of future space robotic systems;
- two prototypes implementing the architecture for two separate (simple) case studies; and
- a methodology for the implementation of GOAC controllers.

One of the areas of possible future development could be in the line of enriching the GOAC framework with additional development and monitoring tools, in order to make the instantiation process of a controller more systematic.

GOAC allows a mission to be specified by means of goals such as “do survey X”. A mission goal consisted of some tens of actions, such as let the robot go to some waypoints, take some pictures, communicate the pictures to another system according to some communication windows, take a sample with the drill, and look for a chance to do opportunistic science. The controller of the robotic system decides how to do it This is the foundation for goal-oriented operations.

It is possible to define goals at different levels of abstraction. Furthermore, the 3DROV case study has shown that GOAC is compatible with all autonomy levels, from E1 to E4.

The implementation of GOAC involves the integration of existing software products. Some of these products had already worked together, but others came from absolutely independent origins; to make them work together has been a non-trivial task. We demonstrated that this integration was feasible and worthwhile.

On-board planning capability is an essential for a goal-oriented controller. The timeline-based approach (APSI) followed in GOAC enables the controller to produce temporally flexible plans. Temporal flexibility allows reacting according to environmental or internal events and accommodating new goals. The latter requires the capability of replanning, which has been demonstrated in both case studies in what has been called opportunistic science.

The planning task can be allocated to separate planners for the sake of scalability. This has been demonstrated in both case studies by setting up several reactors.

In one of the case studies the whole functional layer has been implemented under BIP. In this way, the originally GenoM implementation of the functional layer has been expressed in a formal language (the BIP language), which has allowed analysing it in order to verify the correct behaviour after the integration of the functional layer modules, and adding intra- and inter-module constraints. Such constraints are enforced at run time, which guarantees that they cannot be violated.

In some aspects, the activity has gone beyond the original scope. The use of BIP for plan execution, not described in this paper, has shown a way to verify that a plan is correct when it is dispatched for execution.

Finally, in addition to an architecture, a methodology for implementing GOAC controllers is now available. It is not mature yet, but future developers have tools and techniques for developing autonomous controllers.

The main difficulties in following the GOAC approach lie in the complexity and heterogeneity of the framework. Of course, the problem to be solved is itself complex. Several improvements could be proposed in order to make it easier for future developers to implement GOAC based controllers.

7. ACKNOWLEDGEMENTS

GOAC was developed under the ESA TRP contract “Autonomous Controller”.

We are very thankful to the persons and institutions who contributed to this project: Amedeo Cesta, Simone

Fratini, Andrea Orlandini and Riccardo Rasconi from ISTC-CNR; Félix Ingrand and Lavindra de Silva, from LAAS-CNRS; Saddek Bensalem and Tesnim Abdelatif from VERIMAG; and Kanna Rajan, from MBARI.

8. REFERENCES

[1] ECSS - Space engineering - Space Segment Operability Standard, www.ecss.nl.

[2] 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.

[3] 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. Intl. Symposium on Unmanned Untethered Submersible Technology (UUST) August 2009. Durham, NH.

[4] 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.

[5] S. Fratini, A. Cesta, R. De Benedictis, A. Orlandini, and R. Rasconi. APSI-based Deliberation in Goal-oriented Autonomous Controllers. ASTRA 2011 Workshop, Noordwijk, The Netherlands, April 12-14, 2011, www.esa.int/TEC/Robotics.

[6] S. Fleury, M. Herrb and R. Chatila. GenoM: 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.

[7] A. Ceballos, L. De Silva, M. Herrb, F. Ingrand, A.Mallet, A. Medina, M. Prieto. GenoM as a Robotics Framework for Planetary Rover Surface Operations. ASTRA 2011 Workshop, Noordwijk, The Netherlands, April 12-14, 2011, www.esa.int/TEC/Robotics.

[8] 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.

[9] 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.

[10] 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.

[11] OpenPRS website: https://softs.laas.fr/openrobots/wiki/openprs

[12] A. Ceballos, S. Bensalem, A. Cesta, L. de Silva, S. Fratini, F. Ingrand, J. Ocon, A. Orlandini, F. Py, K. Rajan, R. Rasconi, M. van Winnendael. A Goal-Oriented Autonomous Controller for Space Exploration. ASTRA 2011 Workshop, Noordwijk, The Netherlands, April 12-14, 2011, www.esa.int/TEC/Robotics.

[13] DALA robot homepage: http://homepages.laas.fr/matthieu/robots/dala.shtml

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

[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] ESA EGOS SIMSAT website: www.egos.esa.int/portal/egos-web/products/Simulators/SIMSAT/