First Steps on an On-Ground Autonomy Test Environment

Pablo Muñoz*, Amedeo Cesta†, Andrea Orlandini† and María D. R-Moreno*
*Departamento de Automática, Universidad de Alcalá
28805 Alcalá de Henares (Madrid), Spain
Email: {pmunoz, mdolores}@aut.uah.es
†CNR – Italian National Research Council, ISTC
00185 Rome, Italy
Email: {amedeo.cesta, andrea.orlandini}@istc.cnr.it

***Abstract*—Thanks to the advances in Artificial Intelligence (AI), and in particular in automated planning & scheduling and execution, goal-oriented controllers have being developed to test degrees of autonomy for robotics systems in challenging scenarios. Despite these efforts, there is a lack of methodology for approaching the design of deliberative systems or the choice of the critical parameters or metrics under which the control systems can be compared during execution or deliberation.**

This paper presents the first initial results of the On-Ground Autonomy Test Environment (OGATE) over a year after its initiation. It is an ESA funded project that aims to facilitate accurate experiments on planning and execution systems for robotics. We present features of an initial instance of such system built to support the GOAC robotic software, an ESA project to demonstrate key concepts in autonomy for ESA missions

I. Introduction

The advances in both hardware and software technologies have been translated in the exponential growth of the available functionalities in many real applications, and in particular, in robotic systems. As a result of deploying advanced robotic systems in unknown and dynamic environments, the control software required to achieve the mission goals shall deal with an important number of constraints. Thus, the advances in the Artificial Intelligence (AI) field seem to be naturally merged with the control of robotic systems in order to allow it to generate long term plans without (or little) human interaction. In this way, developments in planning and scheduling systems, such as task planning [1], [2], CSPs [3]–[5], timelines [6]–[12] and, recently, the efforts interleaving planning and execution [13]–[15] could be very valuable to control robots in dynamic environments with an increasing degree of autonomy.

These AI controllers are the top layer of complex tools that are usually made ad-hoc to control a specific robotic platform to perform determinate missions. Usually, to test and verify the correctness of an architecture, a small set of missions are carried out. Also, some parts of the control system could be evaluated in a standalone manner via particular test beds. But testing the robustness, adequacy and performance for the whole control architecture cannot be easily done; it requires to collect and to analyze relevant data from all the parts of the control system while the test bed covers more cases than the typical scenarios. This is currently an open and interesting problem in which not much work has been already done.

In this paper we present the first prototype of the On-Ground Autonomy Test Environment (OGATE) after one year of its initiation. OGATE is funded by the ESA Networking and Partnering Initiative *Cooperative Systems for Autonomous Exploration Missions*. It aims to provide testing support while developing controller systems for space robotics missions. The OGATE environment would constitute an entry point to investigate these open problems in autonomous controllers as the combination of an engineering effort, identifying the requirements, designing and implementing a general environment to provide testing and verification tools for autonomous controller systems; and a research effort to discriminate the key factors in research on planning, scheduling and execution in order to evaluate the performance of autonomous controllers.

We also aim to provide support for ground segment facility in space robotics missions, hiding the complexity of the controlled system to the user. So, in the paper, the space robotics context is exploited as a real-world scenario, employed to test different solutions for deliberation and execution under the same conditions.

The paper is structured as follows: next section describes a space robotic scenario related to the GOAC project and exploited as a case study. Then, the objectives of the system are presented and, in the following, a brief description of OGATE and its functionality is described. The presentation of an initial deployment of the system and a description of what is a plug-in component for OGATE are given. And finally some conclusions are outlined.

II. A Space Robotic Case Study

Our interest in plan-based autonomy is also related to a recent participation in the Goal Oriented Autonomous Controller (GOAC) [16] project: an ESA effort to create a common platform for robotic software development. In particular, the GOAC effort combines several technologies: (a) a timeline-based deliberative layer which integrates a planner, called OMPS [10], built on top of APSI-TRF to synthesize timelines and revise them according to execution needs, and an executive *a la* T-REX [15]; (b) a functional layer [17] which combines a state of the art tool for developing functional modules of robotic systems (G^{en}oM) with a component based framework for implementing embedded real-time systems (BIP).

The GOAC system allows one to implement controllers in a flexible way, i.e., for each robot or mission a different instance of the T-REX system can be deployed defining various cooperating reactors and their associated interactions, providing a scalable architecture. A T-REX agent is composed by a hierarchy of deliberative reactors. Each deliberative reactor has its own deliberative scope as well as planning latency and look-ahead, in charge of controlling a particular aspect of the mission and, interacting with other reactors sending goals and receiving observations. Then, a reactor exploits a planning system to generate plans and to monitor the execution following a sense-plan-act paradigm for goal oriented autonomy. It allows a divide and conquer approach in which the scope of each deliberative reactor could be refined by other more specific reactors. Firstly, the planning system employed in the T-REX deliberative reactors was the EUROPA$_2$ planning and scheduling framework [7], [18]. For the GOAC project the planning system was replaced by the OMPS planning system, which exploit the APSI-TRF execution facilities.

Figure 1 represents a possible instance of the GOAC architecture. In the figure appears two deliberative reactors each one with its own planner and model over which to deliberate. These reactors are interconnected between them, and also, with a command dispatcher reactor. This one is in charge of sending commands to the functional layer (generally composed of different functional modules), while retrieving the observation and propagating them along the other reactors. With these data, the different deliberative reactors, using their respective deliberation models, could dynamically adapt their plans to the new circumstances.

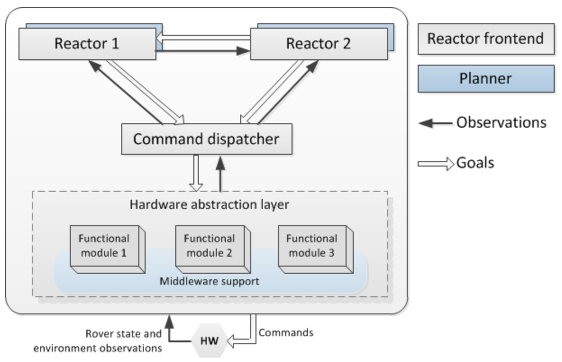


Fig. 1. Representation of a GOAC instance

Within the GOAC initiative, the DALA rover has been considered to simulate a robotic scenario as close as possible to a planetary exploration rover. DALA is one of the LAAS-CNRS robotic platforms that can be used for autonomous exploration 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 this regard, 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.

The considered mission goal is a list of required pictures to be taken in different locations with an associated PTU configuration, and to communicate them to an Orbiter when it is visible to the robot. Also, the rover must operate following some operative rules to maintain safe and effective configurations (the reader may refer to [16] for further details). To deal with the objectives and the operative rules, the OMPS deliberative is in charge of synthesizing a sequence of actions that, starting from the environment state, robot state and goals, reach a final state in which the goals are satisfied.

A possible mission actions sequence is the following: navigate 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. Once all the locations have been visited and all the pictures have been communicated, the mission is considered successfully completed.

Within the GOAC project was also exploited the ExoMars rover model using the ESA 3DROV simulator suite [19] that allows early-stage virtual modelling of terrain and mobile robots systems. The system is composed of multiple modules connected through standardized interfaces, being the most important the *Simulation Framework*, that is the ESA's Simsat, responsible for the execution and scheduling of the simulation and the *Generic Controller* that manages the onboard flight software to enable to connect software modules to control the rover. Also, it includes the *Environment block* in charge of the timekeeping, terrain and atmospheric conditions, and the *Visualization Environment*, a front-end that provides real-time visualization of the simulation progress.

III. ON-GROUND SUPPORT FOR AUTONOMOUS CONTROL

The current investigation is triggered in the space environment where the use of software for on-board autonomy is often perceived as loosing control on critical mission components (namely a space robot, a spacecraft, etc.). For sure current complexity of software for autonomy is quite high and such a complexity reinforce the general skepticism toward its wide use in the space environment. To cope with the problem we have conceived the idea of creating a software environment to be used on-ground to facilitate the demonstration and testing of software for autonomy. Such an environment can also represent the seed for a future knowledge engineering environment for autonomous controllers. Indeed the first goals for such an environment are: (a) facilitating the use of autonomous controllers; (b) allowing the use of different solution for autonomous control (e.g., toward a plug-and-play style in their use); (c) enabling the comparison of different solutions gathering reliable execution data on a given mission. For the time being we are also making an additional assumption: we are focusing our attention toward the deliberative part of the autonomous control the part that can be referred to as the one performing "planning and execution" hence we assume that the physical system is accessible through a functional interface (in GOAC, *a la* G^{en}oM [20]) or through a robotic operating system.

To make our general goal clear let us refer to the space robot domain introduced above. Our research plan is to develop an easy-to-use system able to (i) interface different planning and execution solutions with a same robot (or its simulator) and (ii) to automatically generate realistic test bed scenarios presenting an increasing complexity.

Here, we consider missions related to the space robotics case study. Those missions consist in using a determinate autonomous control architecture over a robotic platform, return some science objectives (i.e., scientific pictures) taking also into account a set of constraints such as the availability time windows for Remote Orbiter communication, or the time period in which science targets are available (some events may be time bounded). The goal is to progressively increase the difficulty of the missions, aiming to stress the planning and execution system and, also, to collect performance information exploiting a real robotic platform like DALA as well as a simulator suites such as 3DROV [19].

Afterward, we are planning to investigate how different ways of conceiving the planning and execution task can be compared. In particular, we need to compare them using the same functional support, as fig. 2 shows, and identical missions.

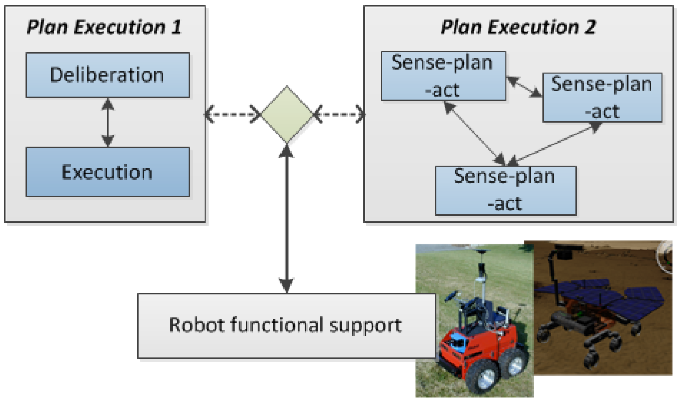


Fig. 2. Different configurations for planning and execution controlling the same robotic platform

Although we can find stress tests for planning systems (for example the International Planning Competition (IPC)[1] in which PDDL-based planners [21] performance is evaluated based on quantitative criteria), they are standalone tests in invariant conditions. This means that there is no interference due to a dynamic environment (e.g., climate changes, external agents, new goal opportunities, etc.) or to changes in the system (e.g., malfunctions or failures) during a mission that affects directly the planning and execution system. Also, newest technologies interleaving planning and execution may use different schemas, for instance, in terms of number of deliberative components or their scope and specialization. Then, a solution plan can also be found by different cooperating autonomous systems. This entails also the need of investigating how these schemas potentially affect the planning process. And, at the best of our knowledge, a methodology to compare performance metrics in realistic scenarios, such as the space robotic mission domain considering not-nominal conditions, is still missing. So, this constitutes an interesting open research issue.

[1]The IPC is usually co-located with the International Conference of Automated Planning and Scheduling (ICAPS).

IV. The OGATE infrastructure

We can briefly define a mission as a set of goals that are solvable by an autonomous agent. This autonomous agent is composed of two parts: i) the platform, which could be a simulator or a robot, and, ii) the control architecture to manage the platform, a set of software components that work together to accomplish the objectives defined in the problem. So, considering an autonomous agent, one or more of these components will be a deliberative component, which are responsible of managing the long term planning to accomplish the objectives of the mission.

Currently, there exists a high number of technologies available to define problems for autonomous agents and their corresponding model of the world (usually called domain), which produce high level plans, which must be decomposed in order to be executed by a robotic platform that accepts low level commands. To deal with the complexity of operating the platform and controlling the execution and decomposition of the high level plans, the control architectures are typically structured in three levels [22]: i) the deliberative layer formed by one or more deliberative components in charge of the long term planning and scheduling ii) the low level support, called functional level that controls the platform using low level commands, and iii) the executive layer, an intermediate level that decomposes the actions produced by the deliberative layer into commands accepted by the functional level. Newer developments interleave the two upper layers (deliberative and executive) into a decisional layer, in which planning and execution are highly coupled.

So, if we are interested in evaluating the performance of a control architecture, we need to take into account all the components, not only the deliberative components. The configuration of the architecture, that is, the hierarchy built on top of a set of different components and how they are connected, plays a fundamental role: some planning technologies generate a complete plan before execute it, while others generate partial plans, interleaving planning and execution in a loop. This implies questions such as the different delays interchanging data between layers that affects the performance of the system. To cover that issue we take the functional support for the robotic platform to control as an invariant part of the system, while employ different technologies for planning and execution over the same domain.

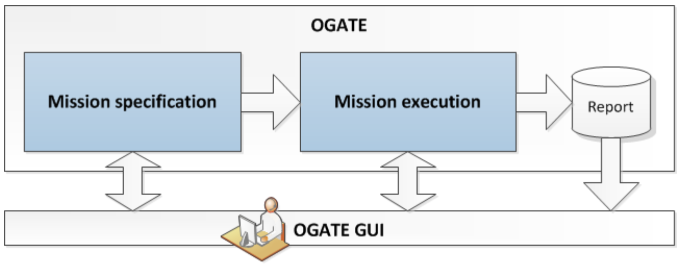


Fig. 3. The OGATE environment

To deal with these questions, OGATE aims at providing an environment to test features of goal oriented controllers and to obtain quantitative comparison based on accurate experiments and also qualitative analysis allowed by inspection and visualization of software internal monitors of the controlled system. The infrastructure of the system relies on three main modules as seen in fig. 3 to support a general test bed environment for autonomous agents:

- **Mission specification:** to define the functionality and goals, first it is required to specify the configuration of the mission: the components of the control architecture and the platform over which they operate. In this way, the system provides a convenient mode to allow the user to configure some components of the controlled system, such as the deliberators. Typically, these AI controllers work with a domain and a problem. First one defines the interactions between different elements of the world, and the last one includes some initial facts and the desired objectives of the mission. The specification of a mission testbench could consist of an evaluation of different control architectures or various configurations for a control architecture to select the best one for a particular mission, or the evaluation of the performance for a particular control architecture over a set of missions, to evaluate and improve the components employed. The OGATE system will be able to support these tests in an automated way.
- **Mission execution:** the mission specification includes the configuration of the different components (executives, deliberatives, etc.) involved in the mission. The execution support of OGATE provides a framework to deal with the complexity of the underlying architecture of the different components, to execute the user defined mission and to gather the relevant data in order to give it to the user. Also, during the execution, the user must be able to interact with the controlled system, modifying internal parameters or including new mission goals to change the nominal execution, in order to test the robustness of the system and the replanning capabilities of the planning and execution components, or to include new science opportunities for real missions to maximize the science return. So, using specific defined interfaces, OGATE is able to access to the different components to control and gather the relevant data.
- **Report:** from the previous data, a research effort to identify and to develop useful metrics for comparing the performance of different deliberative layers will be addressed, in order to obtain strong conclusions when comparing different deliberative components under the same conditions. OGATE will provide a human-legible report of the mission when the required tests have been performed.

V. A FIRST CONTACT WITH THE ENVIRONMENT

For the first deployment of the OGATE environment we employ the GOAC architecture. The mission specification relies on the OGATE infrastructure to provide the configuration of the mission by the user, that is, select a domain and a problem, and the deliberative components that will be attached to the GOAC architecture. Also, it will be possible to define different platforms to control, with their respective functional layers. The configuration of the different reactors for the architecture (i.e. command dispatcher or functional layer) will be easily set by the user using a provided graphical interface, encapsulating the complexity of dealing with the underlying GOAC architecture.

The mission specification generates a configuration file, to be used by the mission execution. The mission specification module manages this configuration allowing reading and generating new configurations. To do this, a XML configuration file is employed. A typical configuration can be see as a tree in which every component provides a functionality while requires (or not) the functionality of other components. Figure 4 shows a valid configuration for the GOAC controller, while fig. 5 presents the hierarchical decomposition of the functionality of each component.

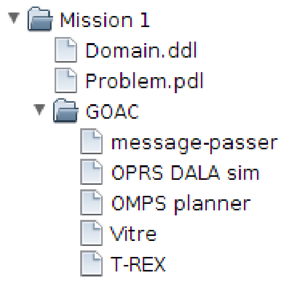


Fig. 4. GOAC mission specification

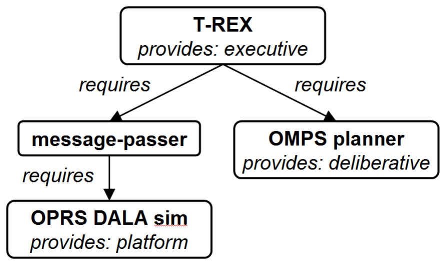


Fig. 5. GOAC functionality hierarchy

The mission execution takes the configuration defined by the user and it attaches the different components to the OGATE system to execute them in a coordinated way. Some of these components are basic components over which OGATE has no control, it only executes them because they are required (for example, a simulation platform will be executed before execute the functional layer that controls the platform), while other components will be controlled directly by OGATE, such as the deliberative component. These components will be defined as OGATE plug-in components (more details in next section). For deliberative plug-in components a GUI allows the user to include new goals or to modify the internal state during the test, showing the relevant data in real-time.

From the data gathered by the plug-in components and other relevant data generated by the system, OGATE will provide a report to measure the performance for these components and for the whole configuration.

At this moment we are focused on the deliberative capabilities, and for the initial test we are interested in two points:

– Starting from the timelines approach and with the APSI-TRF support, deploy an automated problem generator and to perform some initial tests to evaluate the planner, increasing the problem hardness. This generator must be based on critical factors that affect directly to the goal oriented controllers, and the problems must be defined using a set of parameters which denote the problem hardness with valid and objective criteria.

– When the system is mature, we plan to interchange the configuration and technology employed for the deliberative and executive process, using different instances of GOAC for interleaving planning and execution, and other solutions based in different paradigms, such as the *plan-then-act schema*, like MOBAR architecture does [23].

A. OGATE plug-ins

To achieve the goals of OGATE, some parts of the control architecture under study shall be accessible to the system. To cope with this, OGATE defines a small set of easy interfaces which allow controlling different aspect of a component. A OGATE plug-in is a component of a controller system that implements some functionality accessible through these interfaces, giving some access to OGATE to control its execution, or to retrieve data from it. The interfaces that OGATE defines are:

– **Control interface:** provides the basic functionality to run, pause or stop the component safely. Also, this gives a channel to monitor the health of the component, allowing detecting non-nominal states of a controlled component.

– **Data interface:** supplies a bidirectional channel to retrieve the relevant data and to modify the internal state of the component. For deliberative components it also provides a function to include new goals during the execution. This implies that the Data interface acts as a telemetry/telecommand system.

– **GUI Interface:** it is possible to include a specific user interface for that component in the OGATE GUI.

In order to implement the Control and Data interfaces it is possible to select between two interfacing methods:

– **Direct method invocation:** a set of Java interfaces is specifically designed to work with OGATE. This gives an easy to implement synchronous communication between OGATE and plug-ins that can be deployed for components implemented in Java and C/C++ (using the Java Native Interface).

– **OGATE server:** a plug-in can be connected through the OGATE server using a simple protocol supported via TCP/IP. This allows deploying OGATE plug-ins in different computers using an asynchronous schema.

Both methods provides the same functionality and can be mixed, however, using the OGATE server it is also possible to support telemetry in real time, which is also useful to retrieve temporal metrics (see next section). Finally, the GUI interface is only accessible using the direct method invocation.

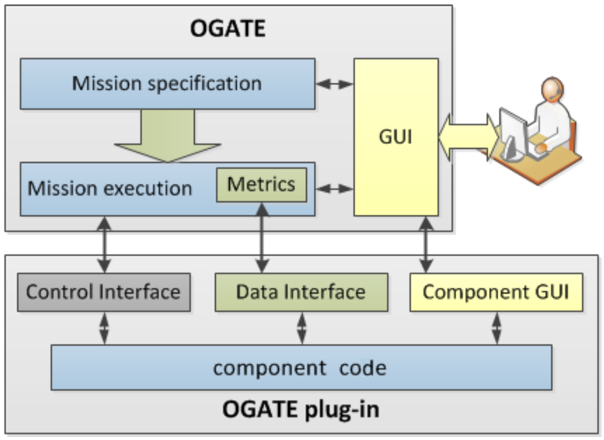


Fig. 6. A deliberative component plug-in

OGATE can integrate a Java GUI inside its environment with a minimum modification of the previous code; graphical components developed in other technologies must be executed outside the environment at this moment.

B. Metrics

From the different plug-ins attached to the OGATE system, the information gathered through the data interface will be passed to the Metrics module, which takes the relevant data and generates a report based on specialized metrics. This is the final result of the execution of the OGATE system.

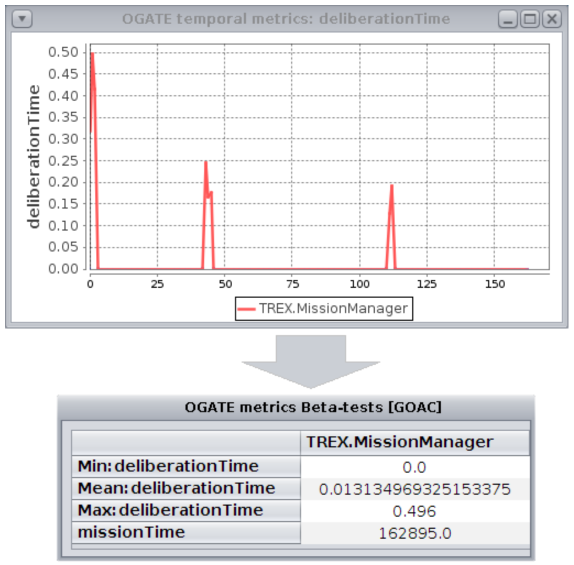
For every mission a set of metrics can be defined. A metric represent a relevant value that we want to retrieve from the execution. The metrics set can be different for each control system and for every component. For example, a common metric can be the time employed by the component during its execution, which can be a metric valid for every component, while a specific metric can be the time spend in searching a solution for a deliberative component.

OGATE allows two types of metrics:

– **Basic:** this metrics are collected at the end of execution, thus, we are only interested in the final value. An example of such metric could be the number of goals achieved by the controller.

– **Temporal:** a temporal metric allows us to keep track of the value for that metric during the whole mission, allowing to see how the value evolves over time. An example of a temporal metric can be the time employed dispatching goals to other components.

At the end of the execution, OGATE will present not only the basic metrics, it also extract relevant values from the temporal metrics. Figure 7 shows an example of a temporal metric named *deliberationTime* that represent the time in miliseconds spend by the component deliberating. During the execution, OGATE presents the chart of the top of the image, while at the end, it present a summary of all metrics defined for the mission. In the bottom of the figure are presented a basic metric, *missionTime*, the time to complete the mission, and, also, the set of values obtained from the temporal metric.



OGATE metrics Beta-tests [GOAC]

	TREX.MissionManager
Min: deliberationTime	0.0
Mean: deliberationTime	0.013134969325153375
Max: deliberationTime	0.496
missionTime	162895.0

Fig. 7. Metrics on OGATE

Currently only extract the maximum and minimum values and the arithmetic mean of the set.

C. Current deploy of OGATE

The deployment of an automated control architecture is usually a complex task, that requires some knowledge about the different components: how are they interconnected, which one must be executed first to be coupled with the others, etc.. So, if we are interested in disseminating our work, we must spend some time in preparing manuals and help new users.

For these new users, when they are able to start the system, a huge amount of data is (generally) generated and presented via the command line or via specialized graphical interface. But depending on the users preferences, they may want to pay attention to some particular data.

The current state of the OGATE system integrates Knowledge Engineering capabilities, using the different components which form the GOAC architecture, what allows us to:

– Create new problems of the space robotic case study domain (mentioned in previous section) using an automatic generator that defines the hardness in function of the number of science tasks and the time in which the Remote Orbiter is visible.

– Create scenarios through a GUI to specify the planning and execution configuration used to solve the problem(s) defined. Currently, the T-REX configuration is fixed to one deliberative reactor, the command dispatcher reactor (connected to the functional support of the robot) and a visualization tool provided by the T-REX framework (it visualizes the current state of the different timelines contained in the domain). So, the user only can change the planning service employed by the deliberative reactor and the domain and problem to resolve.

– Define components that will be run without the OGATE control such as the functional support of the robotic platform or the simulator. Actually in our implementation, the DALA robot simulator is only available.

– Integrate different plug-ins; OGATE not only accepts deliberative plug-ins, other components could be attached to the system, such as the visualization tool of T-REX.

– Attach the different components to generate an instance of the system, which will be executed by OGATE without any intervention of the user.

– Focus on what we are really interesting since inside OGATE is possible to define which components generate relevant data, and then, show only that information inside a unique GUI.

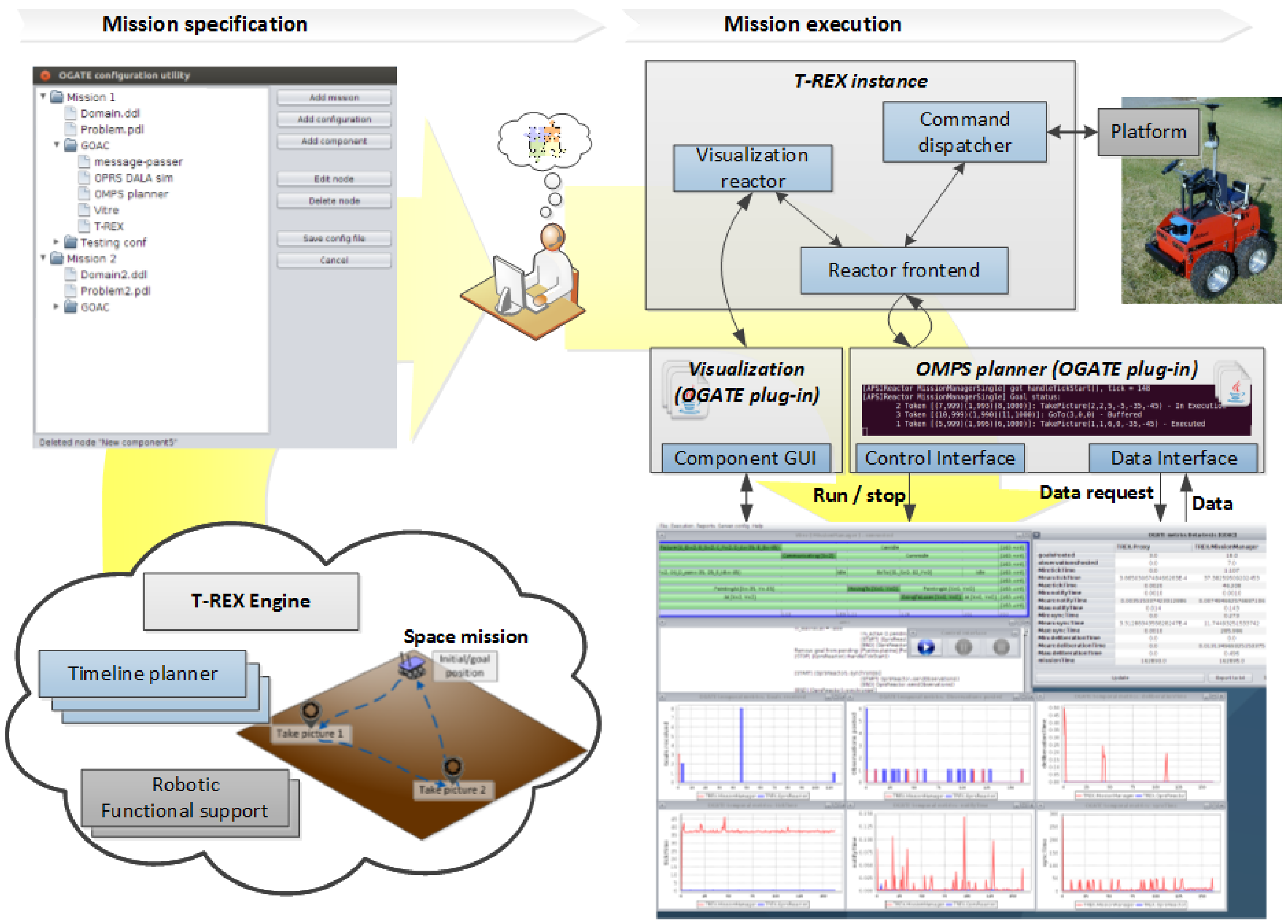
– Display temporal metrics during the mission execution and metrics summary at end.

The cycle for the execution of a control system within OGATE could be seen in fig. 8. Starting from the different components available (the T-REX engine, a timeline-based planner, domain and problem, and a robotic platform), the user can exploit the mission specification module of OGATE to generate a configuration file that defines what are the problems to address, and with which instance of T-REX the problem will be solved within the mission specification GUI (shown at the top left window). After that, the mission execution module takes the user choice and creates an instance of T-REX with the selected planner and the functional support, and it attaches them correctly. In fig. 8 is possible to see (bottom right) OGATE running the current available instance of T-REX with the OMPS planner as the deliberative component. The functional support is the DALA simulator that requires specific components to be executed without being controlled by the OGATE system. One of them is the `mp-oprs`, a message passer service for communication with the robotic platform exploited by OGATE, for which output is presented inside the OGATE GUI. Also, OGATE maintains a registry or log for the different components, which could be valuable for debugging the controlled system. Finally, OGATE must present the relevant data gathered to the user.

In that deployment, the T-REX is modified to support interfacing with OGATE using the OGATE server, and thus, it is possible to retrieve metrics (both temporal and basic) and control the different reactors, which implies the possibility of modify the states of the timelines (to produce failures) or inject goals (for opportunistic science for example). Also, there is another OGATE plug-in to include the T-REX' visualization tool inside the OGATE GUI.

At this very moment OGATE is a system that allows us to interact with T-REX and to obtain some parameters such as the time spend in different tasks for every reactor and other metrics. Starting from this, we want to better understand the interaction between deliberation and execution in order to establish some general metrics that can be useful to analyze and compare different autonomous controllers.

Although OGATE provides some advantages, we are currently working on expanding its capabilities for making execution more "user-comprehensible". At this moment OGATE



```
[APS]Reactor MissionManagerSingle] got handleTickStart(), tick = 148
[APS]Reactor MissionManagerSingle] Goal status:
        2 Token [(7,999)(1,993)(8,1000)]: TakePicture(2,2,3,-3,-35,-45) - In Execution
        3 Token [(10,999)(1,990)(11,1000)]: GoTo(3,0,0) - Buffered
        1 Token [(5,999)(1,995)(6,1000)]: TakePicture(1,1,0,0,-35,-45) - Executed
```

Fig. 8. A first deploy of the OGATE system

shows the information of the components, but not the interaction between them. This is an important issue that has to be addressed to better understand what and how the system works. Also, through the GUI, our intention is to provide a framework that allows the user to support different planning and execution systems via a plug-in style, trying to ease the deployment of complex systems. The support is not only applicable to the deliberative layer, but also to the functional support, in order to exploit different robotics platforms or simulators suites. Finally, our intention is to investigate the metrics to define the performance for planning and execution systems, implementing objective and comprehensive comparison reports for autonomous controllers inside the OGATE system.

VI. Conclusions

Control architectures for autonomous robots are complex software systems, not only from a design and implementation perspective but also from a research point of view since we want to analyze and (possibly) improve their performance. This is in part a consequence of the lack of a methodology to perform test on previous works on autonomous controllers. Also, currently there is no a definition of what are the relevant parameters (metrics) to measure of this kind of systems. So, a testbench that allows the user to define metrics and obtain results from the execution of different tests (by inspection of the relevant parameters) is required as an initial effort to better understand these complex systems and fairly analyze them from a scientific perspective.

Also, it is usually very complex to deploy and execute an autonomous control architecture (which is usually ad-hoc made for a particular robot and purpose), which generally is formed by different interconnected components. Besides, once the systems is executing, the operator gets a huge amount of non human-legible information, confusing the user and keeping him/her away from the relevant data.

For this reason we are working on the OGATE system, which aims to address part of this gap, providing a general testbench to allow easy deployment of an autonomous architecture: giving the OGATE plug-ins and configurations files, it will be easy to deploy any system. It will be only required to load the specific files and run it. Then, specialized graphical interfaces will be enabled to show the relevant data to the user, encapsulating the complexity of the underlying functionality. Finally, OGATE will also offer an automated testbench to obtain quantitative comparison based on accurate experiments, which leads to human-legible reports with well defined metrics.

Acknowledgements

Pablo Muñoz is supported by the European Space Agency (ESA) under the Networking and Partnering Initiative. CNR authors are partially supported by the Italian Ministry for University and Research (MIUR) and CNR under the GECKO Project (Progetto Bandiera "La Fabbrica del Futuro"). Authors want to thank to the ESA's technical officer Mr. Michel Van Winnendael for his continuous support.

References

[1] R. E. Korf, "Planning as Search: A Quantitative Approach," *Artificial Intelligence*, vol. 33, pp. 65–88, 1987.

[2] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem-Proving to Problem-Solving," *Artificial Intelligence*, vol. 2, no. 3, pp. 189–208, 1971.

[3] R. Dechter and J. Pearl, "Network-based Heuristics for Constraint-Satisfaction Problems," *Artificial Intelligence*, vol. 34, no. 1, pp. 1–38, 1987.

[4] M. B. Do and S. Khambhampati, "Solving Planning-Graph by Compiling It Into CSP," in *ICAPS-00. The Fifth International Conference on Artfcial Intelligence Planning and Scheduling*, 2000, pp. 82–91.

[5] A. Cesta, S. Fratini, and A. Oddi, "Planning with Concurrency, Time and Resources: A CSP-Based Approach," in *Intelligent Techniques for Planning*, I. Vlahavas and D. Vrakas, Eds. Idea Group Pubhishing, 2005, pp. 259–295.

[6] N. Muscettola, "HSTS: Integrating Planning and Scheduling," in *Intelligent Scheduling*, Zweben, M. and Fox, M.S., Ed. Morgan Kauffmann, 1994.

[7] A. K. Jonsson, P. H. Morris, N. Muscettola, K. Rajan, and B. D. Smith, "Planning in interplanetary space: Theory and practice," in *the 5th International Conference on Artificial Intelligence Planning Systems*, Breckenridge, Colorado, USA, April 2000.

[8] D. Smith, J. Frank, and A. Jonsson, "Bridging the gap between planning and scheduling," *Knowledge Engieneering Review*, vol. 15, no. 1, pp. 47–83, 2000.

[9] J. Frank and A. Jonsson, "Constraint-based attribute and interval planning," *Journal of Constraints*, vol. 8, no. 4, pp. 339–364, 2003.

[10] S. Fratini, F. Pecora, and A. Cesta, "Unifying Planning and Scheduling as Timelines in a Component-Based Perspective," *Archives of Control Sciences*, vol. 18, no. 2, pp. 231–271, 2008.

[11] A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi, "Developing an end-to-end planning application from a timeline representation framework," in *IAAI-09. Proc. of the The Twenty-First Innovative Applications of Artificial Intelligence Conference*, Pasadena, CA, USA, July 2009.

[12] S. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, and S. Frye, "Timeline-based space operations scheduling with external constraints," in *ICAPS-10. Proc. of the Twentieth International Conference on Automated Planning and Scheduling*, Toronto, Ontario, Canada, May 2010.

[13] J. Ambros-Ingerson and S. Steel, "Integrating Planning, Execution and Monitoring," in *AAAI*. AAAI Press, 1998, pp. 83–88.

[14] A. Finzi, F. Ingrand, and N. Muscettola, "Model-based executive control through reactive planning for autonomous rovers," in *In Proc. of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, September 2004.

[15] F. Py, K. Rajan, and C. McGann, "A Systematic Agent Framework for Situated Autonomous Systems," in *AAMAS-10. Proc. of the 9^{th} Int. Conf. on Autonomous Agents and Multiagent Systems*, 2010.

[16] A. Ceballos, S. Bensalem, A. Cesta, L. D. Silva, S. Fratini, F. Ingrand, J. Ocón, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. V. Winnendael, "A Goal-Oriented Autonomous Controller for Space Exploration," in *ASTRA 2011 - 11th Symposium on Advanced Space Technologies in Robotics and Automation*, Noordwijk, the Netherlands, April 2011.

[17] 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.

[18] J. Bresina, A. Jonsson, P. Morris, and K. Rajan, "Activity planning for the Mars Exploration Rovers," in *in Proc. of the 15th International Conference on Automated Planning and Scheduling*, Monterey, California, USA, June 2005.

[19] P. Poulakis, L. Joudrier, S. Wailliez, and K. Kapellos, "3DROV: A planetary rover system design, simulation and verification tool," in *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, Hollywood, USA, February 2008.

[20] A. Mallet, C. Pasteur, M. Herrb, S. Lemaignan, and F. Ingrand, "GenoM3: Building middleware-independent robotic components," in *2010 IEEE Proc. of the International Conference on Robotics and Automation*, Anchorage, Alaska, USA, May 2010.

[21] A. Gerevini and D. Long, "Plan constraints and preferences in PDDL3," in *Proc. of the Fifth International Planning Competition*, Italy, 2005.

[22] E. Gat, "Three-layer architectures," in *Mobile Robots and Artificial Intelligence*, D. Kortenkamp, R. Bonasso, and R. Murphy, Eds. AAAI Press, 1998, pp. 195–210.

[23] P. Muñoz and M. D. R-Moreno, "Model-Based Architecture on the ESA 3DROV simulator," in *In Proc. of the 23rd ICAPS Application Showcase*, Rome, Italy, June 2013.