IAC-18.D1.6.12x46215

The ERGO framework and its use in planetary/orbital scenarios

J. Ocón [a]*, F.J. Colmenero [a], J. Estremera [a], K. Buckley [a], M. Alonso [a], E. Heredia [a], J.Garcia[a], A. I. Coles [b], A. J. Coles [b], M. Martínez [b], E. Savaş [b] F.Pommerening [c], T.Keller [c], S. Karachalios [d], M. Woods [d] I. Dragomir [e], S. Bensalem [e], P. Dissaux [f], A Schach [f]

[a] *GMV Aerospace and Defense, Isaac Newton 11 PTM Tres Cantos 28760, Spain, Email : jocon@gmv.com*
[b] *Department of Informatics, King's College London, WC2B 4BG, UK, Email: andrew.coles@kcl.ac.uk*
[c] *University of Basel, Spiegelgasse 1, 4051 Basel, Switzerland, Email: {florian.pommerening,tho.keller}@unibas.ch*
[d] *SCISYS UK Ltd, Methuen Park Chippenham SN14 0GB, UK, Email:* mark.woods@scisys.co.uk
[e] *Université Grenoble Alpes – VERIMAG, 2, avenue de Vignate, Gieres, 38610, France Email:*iulia.dragomir@univ-grenoble-alpes.fr>
[f] *Ellidiss Technologies – 24 Quai de la Douane, Brest, 29200, France Email:* Arnaud.Schach@ellidiss.com

Abstract

ERGO (European Robotic Goal-Oriented Autonomous Controller) [1] is one of the six space robotic projects in the frame of the first call of the PERASPERA SRC. ERGO is aimed to future space missions, in which space robots will require a higher level of autonomy (e.g. Exomars or Mars2020). As a framework, ERGO provides a set of components that can be reused and tailored for robots space missions (Orbital, Deep Space or Planetary Explorations) in which the on-board system has to work autonomously, performing complex operations in hazardous environments without human intervention. The concept of autonomy can be applied to a whole set of operations to be performed on-board with no human supervision, such as Martian exploration rovers, deep space probes, or in-orbit assembly robots. In the last decades, the advantages of increasing the level of autonomy in spacecraft have been demonstrated in planetary rovers. At the same time, orbital space missions have already successfully applied autonomy concepts on board, in particular for autonomous event detection and on-board activities planning.

ERGO provides a framework for on-board autonomy systems based on a specific paradigm aimed to facilitate an easy integration and/or expansion covering future mission needs; by using this paradigm, both reactive and deliberative capabilities can be orchestrated on-board. In ERGO, deliberative capabilities are provided via AI techniques: automated planning and machine-learning based vision systems. ERGO also provides a set of tools for developing safety-critical space mission applications and FDIR systems. Moreover, specific components for motion planning, path planning, hazard avoidance and trajectory control are also part of the framework. Finally, ERGO is integrated with the TASTE [2] middleware. All ERGO components are now being tested in an orbital and a planetary scenario.

This paper discusses the ERGO components, its main characteristics, and how they have been applied to an orbital and a planetary scenario. It provides an overview of the evolution of the ERGO system; its main components and the future extensions planned for it.

Keywords: Artificial intelligence, Space Robotics, Planning, Scheduling, Robotic Controllers

Acronyms/Abbreviations

AADL - Architecture Analysis and Design Language
AI – Artificial intelligence
ASN.1 - Abstract Syntax Notation One
BIP – Behaviour, Interaction, Priority
ECSS - European Cooperation for Space Standardization
ESA – European Space Agency
FDIR – Fault, Diagnosis, Isolation and Recovery
GOAC – Goal Oriented Autonomous Controller
GODA – Goal Oriented Data Analysis component
GOTCHA - GOAC TRL increase Convenience enhancements Hardening and Application extension
MDA MDE – Model Driven Architecture Engineering
PDDL - Planning Domain Definition Language
SDL – Specification and Description Language
TASTE - The ASSERT Set of Tools for Engineering
T-REX – Teleo-reactor Executive
URDF- Unified Robot Description Format

1. Introduction

The ERGO project takes on a set of challenges aiming to increase the autonomy of a space robot involved in orbital or planetary exploration missions. The main challenge is **to reach higher levels of autonomy**, understanding autonomy as "the capability of the space segment to continue mission operations and to survive critical situations without relying on ground segment intervention" [3]. To achieve this goal, the framework provides a combined set of software assets, to tackle the specific needs related to space robotics autonomy.

The first capability needed in ERGO is the **possibility to command the system in the form of high-level commands or goals**. A key element for this purpose, is the on-board planner, which is developed based on AI techniques. This on-board planner, the so-called "Stellar", developed explicitly for ERGO, is based on King's College and University of Basel's expertise in previous planners and heuristic techniques, such as Optic [4] or Temporal Fast Downward [5]. By using an on-board planner, the system can be commanded from ground by using high-level goals, while the steps that are necessary to fulfil the goals are derived by the robot itself. The PDDL planning language [6] is used to model both the domain and the problem (high-level objectives to be achieved). The planner's task is: given a set of high level goals, find a sequence of actions to be executed by the robot in order to achieve them.

Another objective in ERGO is **to detect, based on images, serendipitous events that could trigger dynamic replanning**. For this purpose, ERGO includes the GODA scientific agent. GODA is based on previous ESA studies, like PRoViScout [7] or MASTER [8]. GODA is a software component that can be trained to detect objects having a set of characteristics.

In addition, in the ERGO framework, a **scheduling component** is provided: this orchestration role is handled by a robotic main controller (the so-called Agent) implemented by GMV based on the experience and expertise obtained from previous autonomy research programs such as GOAC [9] and GOTCHA [10] . Based on the T-REX paradigm [11] this Agent implements an efficient execution environment for handling different autonomy levels (from single telecommanding – E1 - to goal-commanding – E4). Within the Agent, different control loops are completely coordinated at runtime during the deliberation and execution phases, thus guaranteeing a harmonized control and execution of reactive and deliberative behaviours.

This main controller includes a **generic ground control interface component** that can be tailored for any specific mission.

Furthermore, ERGO provides support for FDIR capabilities, which enhance a robotic system with safe functional autonomy. Given the criticality of such applications, the FDIR components are developed in a rigorous approach based on formal methods. ERGO makes use of BIP [12], developed by Verimag, that has already been successfully applied on different robotic case studies [13],[14]. BIP is a formal language and framework that allows real-time component-based system design. The framework provides different analysis techniques which consolidate the confidence on the system's correctness at different design levels. In ERGO, the BIP tools are extended and applied in a clearly defined process to design, validate, and implement FDIR components specific to each mission.

For the sake of **modularity** the whole ERGO is built based on the TASTE middleware. TASTE is an open source framework, developed by ESA that allows the development of embedded, real-time systems. TASTE relies on key technologies such as standardized modelling languages (e.g., ASN.1 and AADL), code generators and real-time systems analysis, in order to generate the suitable code skeletons and the system executable. Within the ERGO project, we have benefited of the help and support of Ellidiss for the TASTE extensions needed by the ERGO framework
The Ground Control Interface, the Agent, the Planner, as well as the tools mentioned before: TASTE Extensions developed for ERGO and the BIP tools, are part of the so-called "ERGO Core Framework". These are generic tools and components that can be used in any spacecraft robotic application. ERGO also provides components for specific problems, such as motion planning for a robotic arm, and guidance for a planetary rover. In ERGO a dedicated **library for a robotic arm** equipped with a gripper has been developed by GMV; this library that can be tailored to any robotic arm provides high-level primitives to perform pick and drop operations. In addition, a **guidance algorithm** is also available as part of the framework. Developed by Airbus, it is able to perform path planning, trajectory control and hazard avoidance. The component is based on the work performed by Airbus in the frame of the Exomars project. The guidance component can be parametrized to work with rovers of different characteristics.

Figure 1 illustrates the reusable components of the ERGO framework briefly discussed above. Each component is described more thoroughly in the remainder of this paper.

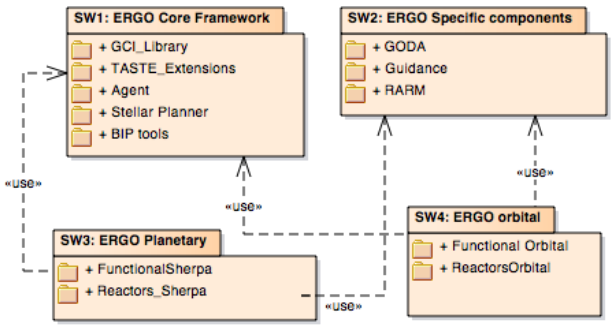

Figure 1: ERGO Framework packages

2. Stellar, the ERGO planner

Stellar is the generic mission planner for ERGO. This novel component is based on concepts and notions from classical planning, timeline-based planning and planning with semantic attachments.

Stellar supports temporal PDDL planning models [6]. The task of planning can be seen as finding a time-stamped sequence of actions that, when executed, transforms a given initial state (such as the current state of the system) into one which satisfies the goals. Fundamentally, this is a combinatorial search problem, with the planner searching over intermediate states along a path to a goal state. In doing so it considers what actions are applicable (according to their preconditions) and their effects on the state. Also, it uses a heuristic estimate to guide search towards the goals. In ERGO, the on-board resource constraints set a tight envelope on the way in which this deliberation can be carried out on-board. Thus, considering the need to perform search, it is important to reduce both the overheads of each state generated during search, and the number of states generated. We will discuss each of these in turn.

Considering per-state overheads, Stellar adopts techniques from the planner Temporal Fast Downward [5], translating the planning problem into a SAS+ [15] multi-valued variable representation. This reduces the memory needed to store each state, by allowing an efficient bit-packed representation – each state variable is stored in just enough bits to cover the size of its domain. To reduce the time overheads incurred for each state, Stellar uses a heuristic that guides the search quickly towards a goal state. We use a tightly coded implementation of the FF heuristic [16], a heuristic that focusses on the causal reasoning that dominates the class of planning problems it is targeting by ignoring delete effects of actions in the heuristic computation. Further, to check that states are temporally consistent (that time windows and other temporal constraints have been met) it uses efficient techniques from the planner OPTIC [4].

Considering the size of the search space in itself, Stellar employs two powerful techniques for temporal planning as forward state-space search. First, it reduces the size of the search space that needs to be considered through the recognition of ‘compression safe’ actions [17]: in many typical cases, this allows a durative action (that can be thought of as having a start, and an end) to be added to the plan as a single step, rather than two steps. Second, it abstracts time window constraints into global constraints applied to all plans, rather than considering their end-points to be plan steps [18].

To allow planning to be coupled with other components of the system, such as Rover Guidance and Robotic Arm (to be described later in sections 6 and 7 respectively), Stellar has an external function interface that allows to incorporate semantic attachments into the planning process [19] . An example of this is shown in Figure 2.

```
(:modules
 (:module ef_orbitalrarmplanner
  (:function (ra_move_dur ?from ?to - slot))
  (:function (ra_move_energy ?to - slot))
  (:function (ra_pick_dur ?from ?to - slot))
  (:function (ra_pick_energy ?to - slot))
  (:function (ra_drop_dur ?from ?to - slot))
  (:function (ra_drop_energy ?to - slot))
  (:function (ra_home_dur ?from - slot))
  (:function (ra_home_energy ?from - slot))
 )
)
```

Figure 2: Example External Functions Definition

At the start of planning, Stellar dynamically loads libraries containing the implementations of the external functions; then it calls functions from the library to determine the values of variables to be used in planning. In the example given in Figure 2, it will load the library ‘orbitalrarmplanner’, and will call the functions defined therein to obtain the values of the given variables (such as ‘ra_move_dur’ – the duration of moving the robotic arm between the two given positions). These variables are used in planning in three ways:

- To determine the duration of actions: how much time will pass between its start and its end)
- To determine the values of variables used in preconditions of actions; for instance, if the remaining energy is sufficient to perform a given operation
- To determine the values of variables used in the effects of actions; for instance, how much energy is used by a given operation.

An important consequence of this external interface is that it allows the planning model to be written at a

higher level of abstraction, where an action encapsulates functionality provided by another component in the system, such as moving the robot arm or moving a planetary rover. Therefore, it suffices that the planner has measures of their time and resource needs without needing a full low-level model of how they will be executed.

To complete the integration of the Stellar planner with the rest of the architecture, a planner 'reactor' acts as an abstraction layer. The reactor generates PDDL models, based on the current state of the system, and contains suitable goals for the current tasks (and any opportunistic science tasks). These are then given to the planner to be solved. The solution plan is then translated by the reactor into a Timeline-based representation [10] to support plan dispatch and execution monitoring. In the case where the observations during execution do not match expectations, the reactor generates a new PDDL model and invokes the planner again to re-plan for this unexpected scenario.

Whilst in theory temporal planning has a high computational complexity **[20]**, Stellar is able to scale well to provide planning capabilities on the specific scientific missions we have considered. Because the on-board resources and capabilities are tightly constrained, the branching factor over what actions could be performed in each state is relatively narrow. This makes the planning task simpler, because there are, in practice, relatively few real choices to be made about how to achieve specific goals. Coupled with an effective heuristic to guide the search for a plan that achieves the goals, and efficient techniques for managing temporal constraints, the planning problems encountered can be solved with modest computational resources.

3. GODA: scientific data analysis

The Goal Orientated Data Analysis (GODA) Component is responsible for processing data from the perception system and generating new candidate goals as input to re-planning activities.

These goals could vary from flagging particular data as pertinent, to directing attention of higher resolution imagers to capture serendipitous science, or perhaps triggering re-planning in order to acquire images from a better position. The key intuition is that intelligent analysis of the environment the system finds itself in may lead to new goals and actions to be taken, which require immediate action and therefore do not allow ground-in-the-loop operations.

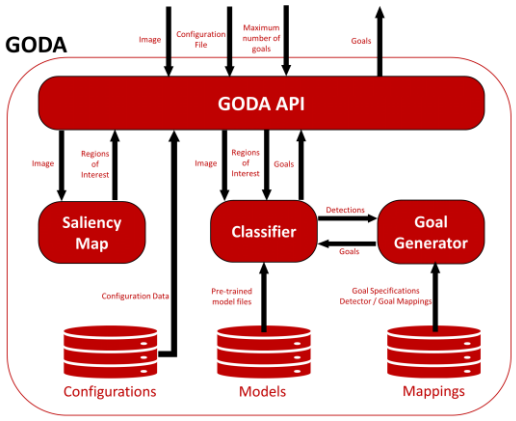

Figure 3: GODA Design Overview

Figure 3 shows the seven main elements of the GODA system. Of these, the Saliency map, the Classifier and the Goal Generator components form the core. The Saliency Map component is designed to segment the image into regions of interest, which the Classifier then classify as known labels. The Goal Generator then maps these detections to specific goals (for the planner) with attached metrics to evaluate. The API forms the interface between GODA and the rest of the system and manages the operation of the core components of GODA.

The Mappings, Models and Configurations files provide the adaptability of the system to different use cases. The models contain the learnt parameters for machine-learning based vision systems implemented in the Classifier. Training these is a computationally expensive offline process which, on the other hand, allows for the on-board detector to be of higher performance and easily adapted to different environments. Similarly, the mapping files manage the mapping between detections, goals and metrics and so allows for re-configuration of the system to suit different objectives. The configurations files are general configuration parameters used by the GODA API to configure each of the components. They are used to abstract the implementation of the functions from any hardcoded configuration values and allow for easy parametrisation if needed.

The Saliency Map, the Classifier and the Goal Generator components are separated, as whilst the operation of the Classifier depends on the output of the Saliency Map, both do not depend on the Goal Generator. This modular design also allows us to exploit advances in the detector performance that future projects will produce by avoiding any requirement for the Saliency Map or the Classifier to depend on the rest of the GODA system. The initial baseline for the Classifier is an instantiation of the pipeline produced for the MASTER project.

Internally to the GODA, the API manages the execution of the Saliency Map, Classifier and Goal Generator components, passing the data between them and performing any conversion necessary to pass images in to the Classifier or convert goals form the Goal Generator component. The structure of the architecture with a wrapping GODA API component also facilitates code re-use, allowing us to exploit our rich background IPR in the field of autonomous science by adapting it to the rest of the system.

4. The ERGO agent and its ground interface

In the ERGO architecture, in order to guarantee a consistent execution of on-board activities, a main controller performs the harmonized execution of a set of control loops, also known as reactors. These reactors interface with the functional layer to command the actuators and receive periodically observations from the sensors. The reactors, together with the controller, form the so-called **agent**.

The number of reactors that form the agent depend on the particular robotic application. But each of the reactors that form the agent perform the control loop associated to a specific functionality. For instance, in the planetary rover use case of ERGO both the mission planner and GODA are reactors that are integrated into the agent, meanwhile in the orbital use case there is no need for a GODA reactor (since there is no serendipitous science to be performed)

The agent's controller is responsible of the correct interaction among the different components and all the messages from/to components are handled by its interfaces with the components (reactors). The controller uses an internal clock to discretize the time in ticks. The duration of a tick can be configured. Time constraints associated to actions in the planner (start and end times, duration) are defined in ticks.

Following the T-REX model, the agent uses a state variable representation to describe the evolution of state over time. We call the instantiated history of such state variable evolution over a temporal horizon as timelines [11]. Each timeline consists of a sequence of procedures which encapsulate and describe state evolution; these instantiated atomic entities are known as **tokens**

The interfaces among reactors are based on goals and observations. A goal specifies an action or state desired to be achieved, meanwhile an observation represents a fact, obtained via the sensors, or deduced on-board based on the information received from the functional layer. So, for instance, a high-level goal can be to perform an experiment on a given place during the current sol (which involves a set of low level goals), meanwhile a low-level goal can be going to a new position. Both goals and observations are communicated using tokens, being each token associated to a timeline.

The agent is in charge of passing these messages (goals and observations) across the different components (the so-called reactors) of the architecture at discrete moments in time, synchronizing them, and providing the tick signal, that instructs the different reactors to perform the work required to fulfil the goals that have been posted to them.

To give a particular example, if the GODA reactor detects an interesting rock in a wide-angle image, it may raise a goal to acquire a high-resolution image of that rock. This goal is then posted to the mission planner. If the mission planner reactor is able to find a plan that is compatible with the fulfilment of all the pending goals, the plan will be changed to accommodate these new activities, and the system will execute the required tasks to acquire the image. This interaction between GODA and the mission planner is detailed in Figure 4.

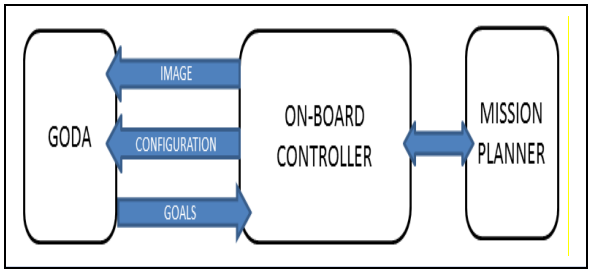

Figure 4: GODA Component interacting with the mission planner via the ERGO controller

As part of the agent, a Ground control interface reactor processes the telecommands and sends back the telemetry. In ERGO, the communication between ground and the spacecraft is based on files. The reason for this is to guarantee consistency and robustness when the latency of the communications is high and there are communication windows (as it is the case in a Martian Rover). Files contain a set of telecommands to be executed, and the model follows a transactional process: either all the commands inside the file are processed or none of them is executed and the file is ignored (this can happen, for instance, if the file is received corrupted or truncated).

The ERGO system has an internal parameter that can be dynamically changed, which defines its level of autonomy. This level of autonomy is defined according to the ECSS standards [3], that is:

- **E1 (direct telecommanding)**: A file is received containing single, low-level commands to be executed immediately.

- **E2 (time-tag telecommanding)**: commands are sent together with a label that identifies the corresponding time for its execution. These commands are executed at their designated time.
- **E3 (Event-driven)**: a plan is uploaded from ground. This plan contains the set of lower-level actions to be executed, together with event-action tables that indicate the commands or on-board procedures to be executed whenever an event is triggered.
- **E4 (goal commanding)**: when running in E4 mode, the system is commanded via high-level goals, and it is the responsibility of the planner to decompose the plan into lower-level actions. The plan is then scheduled and executed by a combination of reactors inside the agent that interface with the functional layer (the so-called command-dispatcher reactors) and a mission planner reactor in charge of verifying that the observations are consistent with the plan's execution. If current observations show that the plan is not being accomplished (for instance, we have not arrived to a given position at a designated time), the agent detects this situation and changes the plan dynamically according to a new solution provided by the planner, based on the remaining high-level goals to be accomplished.

A specific telecommand can be issued to set the autonomy level. In addition, the system is able to autonomously degrade the level of autonomy when there are conditions that could jeopardize the mission (i.e. component failure or lack of a planning solution for a set of goals).

5. FDIR Features in ERGO

In the next generation of autonomous robots, an innovative model-based and dependability-oriented FDIR development approach is required. FDIR components aim to guarantee the safe functioning of a system with respect to desired timed RAMS properties and despite of the errors occurred. In order to do so, FDIR components monitor the events of interest of the system and provide a diagnosis about the occurrence of faults. In case faults have been detected, the components apply functional strategies that bring the system back in a safe state. Therefore, FDIR components extend the autonomy features at the system and mission levels: at system level they allow for a correct functioning and at mission level they enable the attainment of the desired goals.

The approach for designing FDIR components should be supported by rigorous formal methods, providing the possibility of application in the early development stages with short automated development iterations. It should take into consideration the current FDIR architectures and strategies, the development phasing and the schedule constraints for the FDIR development. Additionally, it should allow the effective use of the available software and system designs and the corresponding RAMS analysis data.

As mentioned above, in ERGO we propose and use an approach to design FDIR components based on the BIP tools. We mention that the BIP tools can be applied regardless of the FDIR context for checking the satisfaction of real-time RAMS properties by a system at different levels of the design.

The main approach, implemented in the BIP FDIR tool, consists of the following manual and automated steps:

1. Design the system including both nominal and faulty behaviour as a BIP model. The nominal behaviour describes the system's execution when the environment assumptions are satisfied. The faulty behaviour describes the actions the system executes when faults occur either internally or due to the non-satisfaction of environment assumptions. It is common practice that the two behaviours are obtained as separate models. Therefore, the BIP FDIR tool implements a model merging algorithm generating an extended model on which the FDIR analysis is done.
2. Design the safety property of interest in BIP. This property will be checked by the BIP FDIR tool for satisfaction on the nominal model. If the property does not hold, the system design should be refined by the user based on the provided counter-example.
3. If the property holds, the faults of the extended model that invalidate it are computed as a fault tree. This step, known in the literature as safety analysis/assessment, is automated in the tool.
4. Check that each of the computed faults is detectable given the partial observability of the system, i.e., the diagnosability condition. By partial observability we understand monitoring only a subset of the system's actions sufficient to detect faults, which has important implications with respect to resource consumption. If this automated check fails for a fault, the system design should yet be refined by the user.
5. If all faults are diagnosable, the tool proceeds by synthesizing a diagnoser for each of them. The diagnoser is the fault monitoring part of the FDIR component, given again partial observability.
6. Design the recovery strategies in the BIP model. These correspond to the controller part of the FDIR component, which aims to bring the system back in a safe state. Then the tool produces the full

model containing the extended model and the FDIR component, and validates the behaviour of the controller.

7. If the validation is successful, the C++ implementation of the FDIR component is generated with the BIP compiler and engines. This implementation can be integrated and deployed with the system.

For further details about the approach and implemented algorithms the reader is referred to [21]. We mention that this tool is targeting event-based safety properties. For data-based safety properties, an alternative approach based on BIP validation tools is proposed below. The same approach is used for the ERGO uses cases, and more generally in any use case, to validate the satisfaction of real-time RAMS requirements on a system with the BIP tools.

An alternative approach to design FDIR components, depicted in **Figure 5**, has been used in the ERGO use cases. This approach consists of manually modelling the FDIR component and validating it with the iFinder/iChecker, BIP compiler and engines, and SMC-BIP tools. The reason is that the FDIR components in the ERGO uses cases enforce properties based on data values which would require adding components to transform the property from data-based to event-based. This step would be equivalent to modelling the diagnoser part.

On the full BIP model the user can run the iFinder tool to compositionally compute (an abstraction of) the system's reachable states as an invariant. With iChecker, the user can verify whether a required safety property holds on the system design represented by the invariant. These safety requirements can cover the nominal behaviour, but also the FDIR component behaviour. The BIP compiler and engines generate C++ code from the BIP model. The code is executable, and the system's real-time RAMS can be validated by simulation: interactive, stepwise, real-time, etc. A more thorough validation can be achieved with the SMC-BIP tool. This statistical model-checking tool runs a relevant number of simulations (based on confidence parameters) and checks the probability of satisfaction of the requirements. The tool answers two types of questions: quantitative – what is the probability for requirement satisfaction, and qualitative – is the probability for requirement satisfaction above/under a threshold. In the ERGO use cases we have used SMC-BIP to validate the behaviour of the FDIR component by checking quantitative questions and expecting a probability of around 100%.

Finally, the BIP tools are integrated in the ERGO frame via TASTE. A model transformation from TASTE to BIP is implemented and seamlessly integrated in the TASTE GUI. The generated C++ FDIR component can be easily integrated in a TASTE design by writing the suitable communication wrappers.

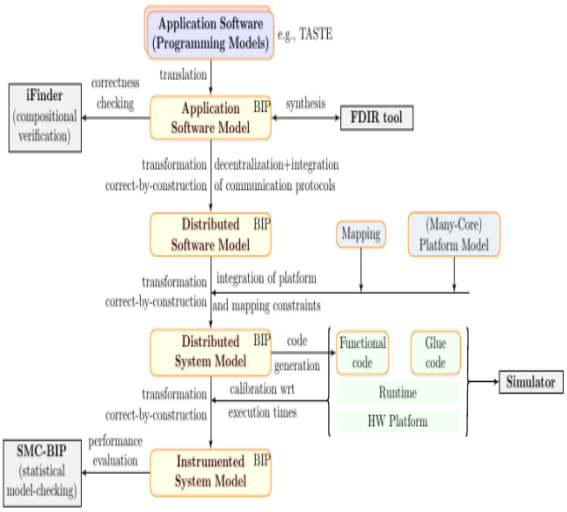

Figure 5: System design process in the context of the BIP framework.

6. Rover Guidance

The Rover Guidance (RG) module is aimed specifically to tackle autonomous guidance of planetary rovers. Guidance includes navigation, path planning, trajectory control, and hazard avoidance.

The ERGO RG architecture is designed to utilise orbital and locally sensed terrain characteristics to maximise the travel distance in function of the traversed terrain difficulty. Different modes of guidance are defined based on the difficulty of the terrain. The RG mode is selected autonomously in function of the terrain difficulty being traversed. This approach is inspired by the ExoMars design studies [22] and NASA rovers that have been operated over the years [23] : using many different driving modes with various levels of functionalities depending on the surrounding terrain being observed. The ERGO Rover Guidance provides:

1. A framework to enable utilisation of various GNC modes suitable for terrains with various difficulties;
2. An autonomous switching between the GNC modes, as needed whilst progressing during the traverse;
3. Building blocks required by all the GNC modes to provide long term guidance and rover safety (see Figure 6).

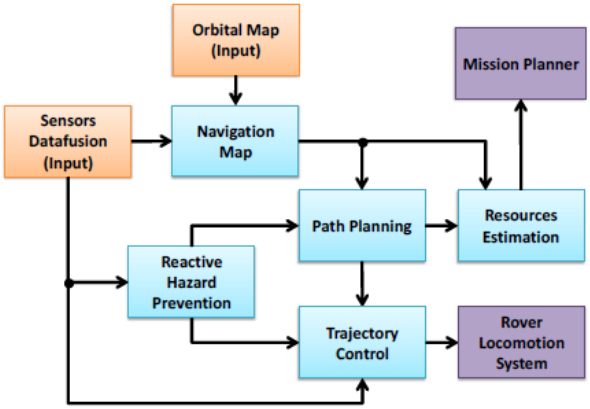

Figure 6: The rover guidance includes (in blue): building of a navigation map, path planning, hazard prevention, trajectory control and resources estimation

The global navigation map created from orbital data contains the level of terrain difficulty per terrain area, referred as "tiles" (i.e. square areas). Each difficulty level is associated a Rover Guidance navigation mode.

The modes are designed as layers, where the higher mode level encompasses the functions of the lower modes, as seen on Figure 7. The higher the mode number, the more planning, mapping and complexity is included, and therefore the computation takes longer leading to slower rover progress.

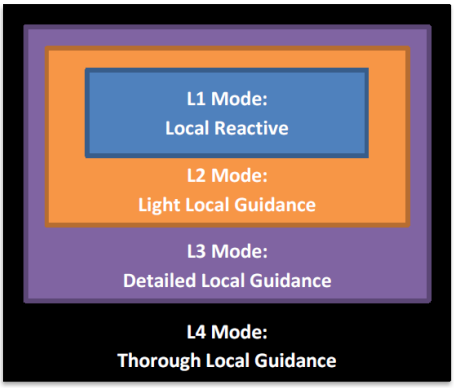

Figure 7: Possible Rover Guidance Modes which incrementally incorporate more complex functions

The transition between the modes is performed autonomously depending on the tile category. Furthermore, according to events while driving, the tile can be re-categorised to a different level of difficulty

7. Robotic arm

The robotic arm component is a general purpose library that can be tailored to any kind of robotic arm. The library is to be used in combination with a set of low-level classes that contain the particularities of the arm. These lower-level classes can be generated based on a URDF [24] model of the robotic arm, by using an URDF parser.

The library consists of two different, functional blocks:

- A planning component generates sequences of commands to be sent to the device in order to achieve a given heading and position. Our library is based on components of the OMPL (Open Motion Planning Library) which implements various path planning algorithms in a generic and efficient way. In particular, the algorithm being used is the RRT Connect algorithm [25] in the joint space of the arm
- An executive component is in charge of controlling the trajectory of the robotic arm. This executive component can include data from a camera, which is received in the control loop to adjust the position of the end effector.

8. The middleware: TASTE

TASTE refers both to the middleware used in the ESROCOS project and to the development process and supporting tool-chain that is used to generate it. ESROCOS [26] is the outcome of the first Operational Grant in the frame of the PERASPERA SRC, aiming the development of a Robotic Operating System able to be used in Space.

Since all the Operational grants of the PERASPERA SRC from the 1st call are to be used combined in the applications developed in the frame of the 2nd PERASPERA call, the design of ERGO is done in such a way that its components are provided as TASTE components. By using TASTE, designers implement their embedded systems using a set of views, abstracting the user from the implementation details of the underlying platform (e.g., real-time operating system, hardware drivers) and guaranteeing the fulfilment of real-time properties. TASTE is being developed by the European Space Agency together with a set of partners from the European space industry and academics. TASTE follows a MDE approach: from a set of Views (Data View, Interface View, Deployment View) code is generated for the main components (the so-called TASTE functions).

The main modelling steps that must be followed to build a software application with TASTE are described below. They are supported by a set of graphical editors.

The DataView expresses all the data types that can be used for communication links occurring between TASTE functions. The DataView is formalized using ASN.1 to benefit from the advanced verification and processing tools that are available for this language. In particular, this choice enables automatic generation of

binary data encoding and decoding protocols that are required for portable and optimized communications.

The Interface View provides an architectural representation of the various subsystems and of their corresponding interfaces and connections. Each subsystem is represented by a TASTE function. All the communications between the TASTE functions will be managed by the TASTE run-time middleware, thus ensuring that the real-time requirements are fulfilled. The language that is used to express TASTE Interface Views is standard AADL.

The Deployment View aims at describing the hardware execution platform onto which the software must run. It mostly consists in selecting a set of hardware elements, such as processors, busses and devices among those proposed by the TASTE hardware library. Doing so ensures that all the required glue code will be properly generated by the TASTE build process and that all the executable files will be produced in case of a distributed software application. The AADL language is also used to represent TASTE Deployment Views.

Once these three views have been completed, they can be processed together to build the software application. This stage is mostly automated and requires few user interactions. The build process involves the following main steps:

- The generation of the Concurrency View is a model transformation that produces a complete AADL real-time model from the three modelling views that have been elaborated by the designer. This AADL model can be used to perform early timing analysis and simulation using any AADL compliant tool, and to adjust real-time attributes if needed.
- The generation of the software applicative and middleware glue source code. This step uses the freshly generated AADL Concurrency View and the various source code fragments that were earlier associated with the DataView and Interface View, to produce and compile the executable files(s).

In ERGO the functional blocks of our architecture are embedded into TASTE functions. The following figure shows the Interface View in TASTE for the robotic arm

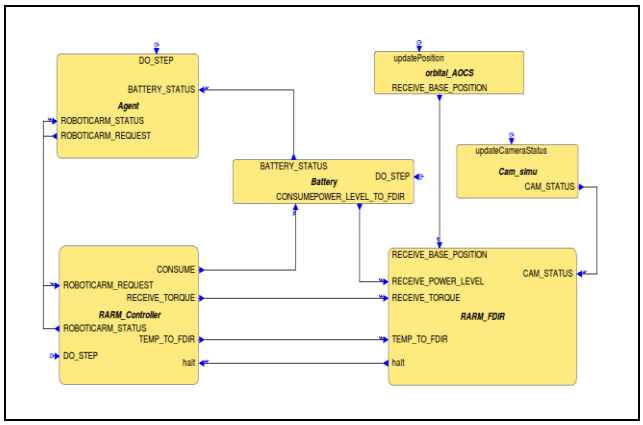

Figure 8: Design of a robotic arm for an orbital mission using TASTE

In the frame of ERGO, a set of improvements were performed to TASTE, namely TASTE editors improvements and prototyping: for time and space partitioning support (TSP) and for BIP model generation via a model transformation from the AADL and SDL models associated with the TASTE functions.

9. Testing and validation. Use Cases

The testing and validation of the ERGO framework has been performed based on a set of layers. Each of the components described in Section 1 to 8 of this paper was contained in a separate repository, and had a single company responsible of it. For each component, unitary tests have been performed. Since some of the components use existing code, the unitary testing has been adapted to the degree of maturity of the existing SW.

In addition to unitary testing, integration tests were performed. These tests helped to see the problems of interaction of different components. The ERGO architecture facilitated this task, since it is designed in such a way that is it relatively easy to add new components when they are available. The following list shows the test accomplished for each component. The lines of code provided can give an idea to the reader of the magnitude of the project. Note that we are considering only the components of the ERGO system, and that we have excluded from the list the TASTE toolset, as well as the code generated in ERGO specifically for the use cases (SW3 and SW4 packages). The TASTE toolset has been developed for many years and it contains a huge number of code lines.

Table 1: KLOC and IT tests for each ERGO component

Component	KLOC	ITs
Agent	50	12
FDIR and BIP tools	350	03
GODA	6.8	02
Guidance	100	04
Planner	64	05
Robotic Arm	80	05

Final validation of the ERGO framework is being performed by applying the framework in demonstration scenarios. As of today, ERGO has been applied to two different use cases, an orbital mission and a planetary mission. These are described hereafter

9.1 ERGO Use case: Planetary mission

The planetary mission consists of a planetary exploration rover, able to pick samples with a robotic arm as well as to take images. The reference mission for the planetary track is inspired on the Mars Sample Return (MSR) mission that covers the concepts and requirements of the Martian Long Range Autonomous Scientist, which could be split in the following phases:

1. **Setup Multi-sol operations** (Ground Control): Ground Control configures the robot and uploads the operations, which might be single or multi-sol. Operations received from Ground Control will be based on any of the following described.
2. **Traverse:** The rover must perform a long range traverse, in the range of 10 km, simulated with a maximum range of 1km, from the landing site to the location where the Sample Catching Rover (SCR) is. A command to take an image with a given heading, once the final position has been achieved by the rover, can be implicit in the traverse operation.
3. **Opportunistic Science**: During the long traverse, the rover will be allowed to perform opportunistic science, limited to close fisheye camera of the Sherpa, providing image acquisition which might imply deviations from the original path.
4. **Sample collection**: the Rover can be requested by ground to pick or drop samples at different locations by using its robotic arm

For the test campaign, two different sites are foreseen:

1. At DFKI facilities in Bremen: A pre-testing campaign consisting of two separate test periods in Bremen has been conducted, aimed to verify that the ERGO system is properly integrated and tested for the Morocco analogue scenario
2. At an analogue site in Morocco: The main tests for replanning, guidance and navigation capabilities as well as scientific detection will be tested at this site.

The robotic platform being used is the SherpaTT rover from DFKI. SherpaTT is a 4-wheeled planetary exploration rover with an actuated suspension system developed for high mobility in irregular terrain. The rover is able to use energy efficient wheeled locomotion (in contrast to legged locomotion) to cover long distances, and at the same time to negotiate difficult terrain by dynamically adapting the wheel suspension to slopes and obstacles. SherpaTT has an overall weight around 150 kg. Due to self-locking gears in the four suspension units, the rover is able to cope with high additional payload weights without increasing energy consumption to maintain the current robot's body pose. The following figure shows the SherpaTT rover while being tested at DFKI facilities

Figure 9: Testing ERGO on the SherpaTT at Bremen (courtesy DFKI)

The tests conducted are similar for both the Morocco desert and the Bremen Facilities: a first test is designed for a nominal mission, in which the rover has 3 separate goals for a) taking a sample at a given position b) taking an image at a given position and c) performing a traverse. This test demonstrates that the system is able to build a plan, and execute it. A second tests is aimed for a long traverse. For the on-field tests, we indicate between brackets the number of repetitions.

Table 2: Preliminary and field tests – ERGO planetary

Tests	Preliminary	On Field
Nominal mission	07	03 [3-4]
Long traverse	01*	01 [3-4]
Replanning	01	02 [3-4]
Opportunistic science	00	01 [3-4]
FDIR tests	02	01 [3-4]
Autonomy level tests	04	01 [3-4]

9.2 ERGO Use Case: Orbital mission

The reference mission for the orbital track is the On-Orbit Servicing mission, where a damaged spacecraft can have one of its modules replaced autonomously by a servicer spacecraft.

The scenario will simulate a servicing S/C mission. A servicing or 'chaser' spacecraft approaches to a faulty or serviced S/C the so-called 'target'. The chaser will reconfigure the target S/C, so the target must simulate a modular S/C with some faulty/damaged modules (ATMs) which the chaser will have to replace in orbit.

The objective of this orbital scenario is the evaluation of the autonomy performances, so that the architecture and test environment must allow to demonstrate reactivity to scenario modifications caused by two different sources:

1. Failures such as pieces or tools not present in the expected place or found in a different attitude, obstacles in the visual field, changes in the illumination constraints, failure in grasping pieces etc.
2. Deviations with respect to the nominal mission, such as reconfiguration of the spacecraft due to mission constraints (deadlines exceeded, for instance)

In both cases, re-planning is based on updated information from the environment. For that purpose, feedback information needs to be obtained by passive visual means (camera) and force/torque feedback from the robot end-effecter closing control loops at different levels.

The set of tests designed for this use case is shown in Table 3.

Table 3: Preliminary and field tests – ERGO orbital

Tests	Preliminary	On Field
Nominal mission	01	01 [4-6]
Faulty APM	01	01 [4-6]
Errors while manipulating Rob. arm	01	01 [4-6]
Chaser deviates from target	N/A	01 [4-6]
Autonomy commanding	01	01 [4-6]

10. Conclusions

The ERGO framework was designed and developed having a set of objectives in mind.

1. **Goal commanding**: When a set of high level commands are sent from ground, the ERGO agent is able to dynamically generate plans to achieve these goals, deterministically dispatch the associated activities for its execution and is also able to recover from off-nominal conditions, performing re-planning when needed. Plan representation and execution deals with metric time and resources necessary for dealing with planning and execution time uncertainty in dynamic environments. The lower-level functional layer is tightly integrated with the abstract decisional level embodied in the agent. Reactive and deliberative capabilities are handled in a harmonized fashion.
2. **On-board planning capability.** The ERGO planner, Stellar, combines effectively concepts and notions from classical planning, timeline-based planning and planning with semantic attachments. Stellar has been developed for its use in space systems, in which computational resources are scarce.
3. **Dynamic replanning** in the ERGO system can be triggered either 1) when new goals are received from ground, 2) if the constraints of the current plan are being violated during its execution, or 3) when GODA, our scientific agent, detects an interesting scientific event, that requires further analysis (and therefore changes in the existing plan). GODA can be easily configured to different use cases by loading the learnt parameters for machine-learning based vision systems. GODA can also flag the importance of existing on-board images. GODA allows any future robotic

platform developed in ERGO to perform serendipitous science gathering.

4. **Selectable levels of autonomy**: The level of autonomy is selectable by ground and adapts to the circumstances of the mission. If the level of autonomy of the system is lowered by a telecommand sent from ground, the planning capability is disabled. This allows ground to take control whenever there is any system malfunction that requires further analysis, or under special, hazardous situations. The agent can also autonomously lower the level of autonomy, disabling the planning capabilities, when the planner is not able to find a solution for the current goals, or under special circumstances (H/W errors).

5. **Formal verification and validation**: ERGO is extended with verification and validation features via the BIP tools. The BIP tools allow checking the satisfaction of real-time RAMS properties, which is a critical aspect for autonomous systems. Additionally, the BIP tools allow generating valid FDIR components (implementations) with respect to the system design and requirements to satisfy, thus increasing the functional autonomy of such systems.

6. **Guidance and motion-planning ability**: ERGO provides a Rover Guidance component, for its use in planetary exploration rovers that provides path planning, hazard avoidance, as well as trajectory control. Different modes of guidance are defined based on the difficulty of the terrain. The RG mode is selected autonomously in function of the terrain difficulty being traversed. The Guidance adapts dynamically its computations to the difficulty of the terrain being traversed. ERGO also provides a robotic arm library that can be tailored to perform motion planning and robotic arm control.

7. **Model-driven approach**: In ERGO the TASTE framework is used to define, following a model-driven architecture approach, the interfaces among the different components, integrate all of them into different executables, and deploy them in different nodes. TASTE automatically generates glue code that guarantees the proper delivery and reception of messages according to the interfaces defined in a format that is language-independent (ASN.1). What is also important, the use of TASTE in ERGO guarantees its compatibility with ESROCOS, one of the first space robotics projects in the first call of PERASPERA SRC. ESROCOS is aimed to the development of a robotic operating system. The capabilities of ESROCOS and ERGO will be used in combination in the second call of the PERASPERA SRC, for the development of different robotic applications.

ERGO has been applied to two different use cases: an orbital and a planetary use case. The project is now reaching its last phases, in which the system field tests will be performed. Although we will have to wait until the finalization of these tests for our final conclusions, preliminary field tests have already been executed, and the results show that the integration of the different components for both use cases has been performed successfully. Considering the complexity of the different components, and the difficulty of the objectives to be achieved, we think that the results are very positive. We sincerely expect that this framework will pave the way for future developments in space missions

Acknowledgements

We would like to thank the European Commission and the members of the PERASPERA programme Support Activity (ESA as coordinator, ASI, CDTI, CNES, DLR and UKSA) for their support and guidance in the ERGO activity. We also want to thank Malte Wirkus and Thomas Vögele, from DFKI, for their support in the integration of ERGO's software in the SherpaTT rover.

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730086.

References

[1] ERGO web site: http://www.h2020-ergo.eu/

[2] M. Perrotin, J.-L. Terraillon, C. Honvault Taste: towards a space system development framework, Oct. 2015

[3] ECSS Secretariat. (ESA/ESTEC), "ECSS-E-70-11 Space Segment Operability" (August, 2005)

[4] J. Benton, A. J. Coles, A. I. Coles, Temporal Planning with Preferences and Time-Dependent Continuous Costs, Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2012

[5] P. Eyerich, R Mattmüller, G. Röger, Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning, Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2009

[6] M. Fox, D Long, PDDL2.1: An Extension of PDDL

for Expressing Temporal Planning Domain, Journal of AI Research 20 (2003)

[7] G. Paar, M. Woods, C. Gimkiewicz, F. Labrosse, A. Medina, L. Tyler, D. P. Barnes, G. Fritz, and K. Kapellos, "PRoViScout: a planetary scouting rover demonstrator," Proc SPIE 8301 Intell. Robots Comput. Vis. XXIX Algorithms Tech., p. 83010A–83010A–14, 2012.

[8] I. Wallace, M. Woods, "MASTER: A Mobile Autonomous Scientist for Terrestrial and Extra-Terrestrial Research, 13th Symposium on Advanced Space Technologies in Robotics and Automation, ASTRA 2015

[9] A. Ceballos S. Bensalem, A. Cesta, L. de Silva, S. Fratini, F. Ingrand, J. Ocón. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. van Winnendael, Bensalem, S., , "A Goal-Oriented Autonomous Controller for space exploration

[10] J. Ocón, J. M. Delfa, T. de la Rosa, A. Garcia, Y. Escudero, GOTCHA: An Autonomous Controller for the Space Domain, Proceedings of the Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA), 2017

[11] T-REX: A model-based architecture for AUV control. C McGann, F Py, K Rajan, H Thomas, R Henthorn, R McEwen. 3rd Workshop on Planning and Plan Execution for Real-World Systems 2007

[12] A. Basu, M. Bozga, J. Sifakis, Modeling Heterogeneous Real-Time Components in BIP. In Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM) 2006, IEEE, 2006, pp. 3 – 12

[13] S. Bensalem, F. Ingrand, J. Sifakis, Autonomous Robot Software Design Challenge. In 6th IARP/IEEE-RAS/EURON Joint Workshop on Technical Challenge for Dependable Robots in Human Environments, 2008

[14 A. Basu, M. Gallien, C. Lesire, T. Nguyen, S. Bensalem, F. Ingrand, J. Sifakis, Incremental Component-Based Construction and Verification of a Robotic System. In European Conference on Artificial Intelligence (ECAI) 2008, IOS Press, volume 178 of FAIA, pp. 631 – 635

[15] C. Backström, B. Nebel, Complexity Results for SAS+ Planning, Computational Intelligence 11 (1995)

[16] J. Hoffmann, B. Nebel, The FF Planning System: Fast Plan Generation Through Heuristic Search, Journal of AI Research 14 (2001)

[17] A. J. Coles, A. I. Coles, M. Fox, D. Long, Extending the Use of Inference in Temporal Planning as Forwards Search", Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2009

[18] K. Tierney, A. J. Coles, A. I. Coles, C. Kroer, A. Britt, R. M. Jensen, Automated Planning for Liner Shipping Fleet Repositioning.", Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2012

[19] C. Dornhege, P.Eyerich, T.Keller, S.Trüg, M.Brenner, B.Nebel, Semantic Attachments for Domain-Independent Planning Systems, Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2009

[20] J. Rintanen, Complexity of Concurrent Temporal Planning, Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2007

[21] I. Dragomir, S. Iosti, M. Bozga, S. Bensalem, Designing Systems with Detection and Reconfiguration Capabilities: a Formal Approach. In 8th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA) 2018, Springer

[22] M. Winter et al., "ExoMars Rover Vehicle: Detailed Description of the GNC System", Proceedings of Space Technologies in Robotics and Automation (ASTRA), Noordwijk, The Netherlands, 2015, 11-13 May.

[23] J. J. Biesiadecky et al, The Mars Exploration Rover Surface Mobility Flight Software: Driving Ambition, IEEE Aerospace Conference Proceedings, 2006

[24] http://wiki.ros.org/urdf/XML/model

[25] James J. Kuffner, Jr., Steven M. LaValle "RRT-Connect: An Efficient Approach to Single-Query Path Planning". In Proc. 2000 IEEE Int'l Conf. on Robotics and Automation (ICRA 2000)

[26] M. Muñoz et al; ESROCOS: a robotic operating system for space and terrestrial applications. (2017) In: 14th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2017), 20 June 2017 - 21 June 2017 (Scheltema, Netherlands)