USING THE ERGO FRAMEWORK FOR SPACE ROBOTICS IN A PLANETARY AND AN ORBITAL SCENARIO

*J. Ocón[1], K. Buckley[1], F.J. Colmenero[1], S. Bensalem[2], I. Dragomir[2], S. Karachalios[3], M. Woods[3], F. Pommerening[4], T.Keller[4]

[1]GMV Aerospace and Defense, Isaac Newton 11 PTM Tres Cantos 28760, Spain, Email : jocon@gmv.com

[2]Universite Grenoble Alpes, 700 Avenue Centrale 38400 St Martin D'Heres, France, Email: {saddek.bensalem|iulia.dragomi@univ-grenoble-alpes.fr}

[3]SCISYS UK Ltd, Methuen Park Chippenham SN14 0GB, UK, Email: mark.woods@scisys.co.uk

[4]University of Basel, Spiegelgasse 1, 4051 Basel, Switzerland, Email: {florian.pommerening|tho.keller}@unibas.ch

ABSTRACT

The European Robotic Goal-Oriented Autonomous Controller ERGO [1] is one of the six space robotic projects in the frame of the PERASPERA SRC [2]. Its main objective is to provide an autonomous framework for future space robots that will be able to perform its activities without the need of constant human supervision and control. Future space missions, in particular those aimed at Deep Space or planetary exploration, such as Exomars [3], or Mars2020 [4] demand a greater level of autonomy. The concept of autonomy applies here to a whole set of operations to be performed on-board without human supervision; for instance, a Martian rover has to avoid getting stuck in the sand while traversing, autonomously recharge its batteries periodically, and communicate with Earth occasionally each sol [5] . Additionally, it will need to be able to detect serendipitous events (e.g. a rock that has a specific property). A deep space probe [6] has to take the right measurements to approach an asteroid, and due to the latency of the communication with Ground, these measurements need to be taken autonomously on board. Orbital space missions have already successfully applied autonomy concepts on board, in particular for autonomous event detection and on-board activities planning [7].

In ERGO we provide a framework for autonomy aimed to cover a wide set of a capabilities, ranging from reactive capabilities (i.e. capabilities that demand a quick response) to deliberative capabilities (that consider different courses of actions, and evaluate among the different possibilities the best alternative).

This paper will discuss the process of the design of robotic systems using the paradigm provided by this framework applied to two different scenarios: a Sample Fetching Rover (SFR), and also an On-Orbit Servicing mission, where a damaged spacecraft can have one or several of its modules replaced autonomously by a servicer spacecraft. We will describe the methodology, the main problems found, the design decisions taken to overcome these problems, as well as an overview of the final design of both systems

1 INTRODUCTION

Control architectures form the backbone of complete robotic systems. Complex robotic systems require concurrent embedded real-time performance, and are typically too complex to be developed and operated using conventional programming techniques. The complex demands of such systems require frameworks and tools that are based on well-defined concepts that enable the effective realization of systems to meet high-level goals.

An autonomous software framework represents a type of system commonly known in the literature as a robotic architecture, the backbone of the autonomous robotic software around which the rest of the components are defined. A typical architecture is divided into layers to improve the modularity and cater for the different latencies demanded during the construction of plans and execution. Literature distinguishes among three types of approaches:

Deliberative architectures are a classical approach to building controllers, namely, a particular type of knowledge-based system and is defined to be one that contains an explicitly represented symbolic model of the world. The approach suggests that intelligent behaviour can be generated by providing a system with a symbolic representation of its environment and its desired behaviour and by syntactically manipulating this representation. The core of this deliberative architecture is a planner, which elaborates plans based on the knowledge of the problem domain. A plan defines a series of actions designed to accomplish a set of goals but not violate any resource limitation, temporal or state constraints, or other spacecraft or rover operation rules. But any plan, no matter how it is generated, requires the help of an execution system to be useful for real-world execution.

Reactive architectures are based on the idea that intelligent rational behaviour is innately linked to the environment and the idea that intelligent behaviour emerges from the interaction of various simpler behaviours. In rapidly changing environments, such as the real world, where the controller must react quickly to external changes, there may not be time available to perform many time-consuming actions such as planning and introspection. In this case an agent with a reactive, or behavioural, architecture may be more appropriate.

Layered architectures contain a slower, deliberative reasoning system in charge of deciding the strategic plan and a fast, reactive system in charge of reacting to unexpected changes of the environment. This approach breaks down the different proactive and reactive elements of a controller into different layers. This level of abstraction allows complex controllers to be modelled more easily and is flexible enough for use in many robotic environments. The classical layered architecture found in many systems is a three-layered architecture, in which a deliberative layer receives high goals that are managed by an executive layer that in turn interfaces with the lowest, functional layer.

In ERGO a robotic controller, based on GMV experiences on previous research projects for autonomy (like GOAC [8] and GOTCHA [9]) provides a paradigm for handling autonomy capabilities, in which different control loops are managed during execution by a central agent controller, which guarantees a harmonic execution of both reactive and deliberative behaviours. This architecture is based on previous systems, namely IDEA [10], and T-REX [11].

The ERGO architecture for autonomy is a layered architecture, in which a set of reactive and deliberative control loops are embedded into an agent. The agent as a whole contains the executive and deliberative layers that are found in a three-layer architecture. In ERGO, the system is divided in two main components: the robotic functional layer, and the agent.

Deliberative capabilities in ERGO are provided by the use of an on-board planner. One of the main components of ERGO is a newly developed planning system, Stellar, a PDDL-based planner being developed by Kings College London, the University of Basel and GMV-UK.

Moreover, ERGO is designed having in mind a model-driven-development (MDD) approach. It does so by using the TASTE technology. TASTE [12] is a Model Driven Engineering approach that provides a methodology and a set of tools to build dependable embedded software with real-time constraints. It has been developed as a follow-on of the ASSERT EC-FP6 project, and has been promoted and founded by ESA. TASTE is the middleware selected for ESROCOS [13], which is also a robotic project of the PERASPERA SRC aimed to the development of a robotic operating system able to be used in space. By using TASTE in the ERGO framework we guarantee a strong synergy with both projects, as well as its future compatibility. In addition to these capabilities, ERGO also uses the BIP Framework [14] to integrate nominal and error models, and simulate the behaviour of the system in presence of faults.

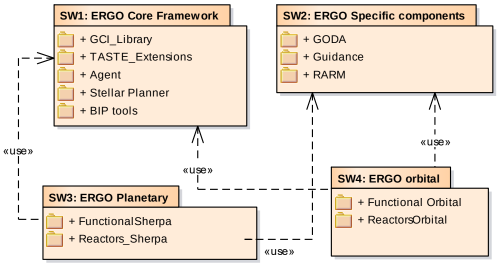

Figure 1: ERGO framework SW packages

2 ERGO SW COMPONENTS

The ERGO framework provides a set of packages and components that can be reused and tailored to develop a robotic platform that can be commanded using any of the 4 levels of autonomy defined in the ECSS Standards [15] , that is:

- **E1 Direct telecommanding** processed as they are received from Ground**.**
- **E2 Time-tag commanding**: commands are set with an associated time-tag, and executed at a designated time.
- **E3 Event-driven**: in which a set of pre-defined events trigger on-board control procedures or direct commands.
- **E4 Goal commanding**: in which the system is commanded via high-level goals that are decomposed by a planner into a sequence of lower level commands to be executed. An executive controls the execution of these low-level commands.

The main packages and their components of the ERGO framework are shown in Figure 1. The ERGO Framework is composed of four main SW packages. The first two packages (SW1 and SW2) are aimed to be re-used across different robotic platforms meanwhile the remaining packages (SW3 and SW4) contain the SW for two specific implementations of ERGO (the so-called ERGO use cases, described in section 3 of this paper). In the following, we will describe the contents of SW1 and SW2 that provide a set of common components developed to be reused across different robotic platforms.

2.1 SW1: ERGO Core Framework

The first and most important package of ERGO is the core framework. The core framework package provides a set of tools to build an autonomous robotic system in any robotic application. It contains the following sub-packages:

Agent: In ERGO a single agent conforms the deliberative and executive layers of a traditional three-layer architecture. This single agent is composed by a controller (a generic component, common for any robotic asset) and a set of mission-specific components. These are the so-called reactors, and interact with the agent's controller using a single interface.

Following the T-REX architecture [11] each reactor is aimed to control a single control loop: for instance, in a Mars-sample rover application a single reactor (the guidance reactor) can be in charge of controlling the movement of the rover, meanwhile another reactor can be in charge of the control of movements of the rover's robotic arm.

Reactors are hierarchically structured, so that on top of these lower-level reactors there can be a planner reactor. This planner reactor is a deliberative component, able to decompose higher level goals (e.g. "extract a sample at a given position, analyse it, and downlink the data when there is communication with Ground control") into a coordinated set of lower-level actions, to be performed by the lower-level components (e.g., involving a sequence of operations for the guidance reactor and robotic arm reactor in order to reach a given position and pick a sample).

At the top of the reactor's hierarchy, a Ground control interface reactor handles the communication with Ground. Depending on the needs of the robotic application, a different set of reactors can be tailored to manage a particular subsystem. Reactors can be either reactive or deliberative.

Therefore the ERGO framework provides an N-layered architecture, in which different control loops (either reactive or deliberative) can be embedded into functional blocks (reactors) that use a common interface to the (generic) agent controller. Reactors that conform the ERGO agent are tailored and adapted to the specific robotic platform.

This agent sub-package provides the framework to be used to build reactors, and to embed and link them together with the controller.

Ground Control interface: A main component of any space application is the interface with Ground. In ERGO the Ground control interface is a reactor in charge of handing Telecommands and Telemetry, in charge of providing services for different levels of autonomy (time-tag commanding for E2, Action-Event service for E3, on-board planning for E4). This reactor is provided as part of the ERGO Framework, and can be easily tailored to any specific space robot.

Stellar Mission Planner: Stellar Mission Planner: In ERGO, we develop a new planner that is specifically designed to take into consideration the requirements for space robotics applications. The input language is based on the Planning Domain Definition Language (PDDL) [24], a standard planning language. We extend PDDL with semantic attachments [25], which define external functions that allow a tighter integration of the Stellar mission planner and the ERGO specific components that are part of SW2 (here, the rover guidance and robotic arm subcomponents).

Stellar is a heuristic forward search planner with several adjustments to the challenges of space robotics:

1. For maximum flexibility and execution stability of the provided plans, we associate each state with a simple temporal network (STN) that allows to describe start times, end times as well as the duration of actions as intervals;
2. States are expanded in a multi-step process to delay the (costly) evaluation of STNs and external functions;
3. States are managed in a way that allows to switch between a fast yet memory-expensive explicit representation of states in memory and a slower but memory-light implicit representation of states via action sequences; and finally
4. We believe that the comparably novel class of potential heuristics [26] is the perfect candidate for a space mission with scarce on-board resources. These heuristics are evaluated in a state as the sum of the feature weights of all features that hold in that state, which takes negligible resources during planning time. Since the computation of accurate weights is very expensive, we plan to perform this step on Ground before the mission starts and upload updates only when required.

BIP Tools: The ERGO framework is extended with formal verification and validation capabilities for reliability, availability, maintainability and safety (RAMS) requirements. The aim is to ensure correct-by-construction design of safety- and mission-critical systems subject to faults and failures, such as robotic ones. This feature consists of two (offline) tools developed by Verimag based on the BIP framework [14]:

- **iFinder**. This tool applies a compositional verification technique based on invariants to check the satisfaction of safety requirements for timed systems.
- **FDIR BIP tool**. It applies synthesis techniques (i.e., a correct-by-construction approach) for obtaining FDIR components from the system design, requirements to ensure and recovery strategies.

2.2 SW2: ERGO Specific Components

This package consists of a set of subcomponents aimed to solve specific problems of space robotics applications:

GODA: Most robotic applications require a specific module or functional block able to identify patterns from images taken by the robot. In ERGO, this functionality is provided by a separate package named GODA **(Goal Oriented Data Analysis).** This is a SW package developed by SCISYS, able to be used to build scientific detection reactors (components), capable of detecting serendipitous events. In the ERGO architecture, once a serendipitous event is detected by GODA, the GODA reactor is able to post new goals to the planner (i.e. to take an image of this particular area) whenever an interesting event is raised. More precisely, GODA processes data from the sensors and generates new candidate goals as inputs to the re-planning activities. The GODA component of ERGO builds on work from several previous ESA studies developed by SCISYS, in particular, the MASTER project [16] .

Rover Guidance (RG): Planetary exploration rovers require a set of specific autonomous functionalities to enable safe traverse. This includes assessing the traversability of the terrain, planning short and long term path, detecting and avoiding potential hazards, as well as controlling the rover trajectory. This is provided by Airbus, building on their experience in the ExoMars Rover mission [17]. The RG also estimates the resources that the rover requires to perform a long traverse, which can be used to inform the on-board planner. The RG is specifically designed for fast and optimal traverse over long distances and can be tailored for different rover platforms.

Robotic arm: Finally, most robotic platforms are equipped with a robotic arm. For these, ERGO provides the RARM sub-package that contains a set of components in charge of planning & executing the movements of the robotic arm. More precisely, these components allow to plan and execute trajectories and paths between points without any collision. As in the previous cases, the corresponding SW can be reused across many different platforms. The robotic arm component is developed by GMV.

3 INSTANTIATING ERGO

3.1 The Instantiation Methodology

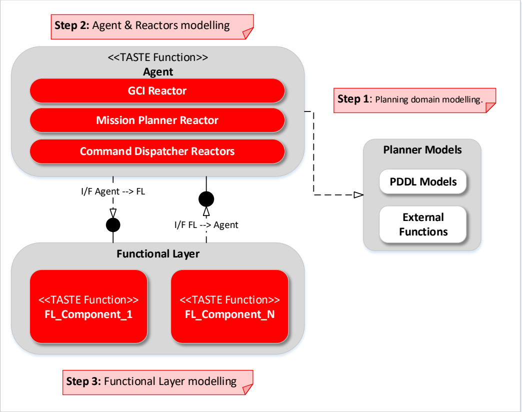

Figure 2: Instantiating ERGO for a robotic platform

The instantiation of ERGO is a process that involves a number of design and implementation tasks and design decisions. ERGO relies on three different components that need to be tailored for each mission, and which are shown in the figure below:

Step 1: **Definition of the planner models**: domain and problem PDDL files that describe the planning model as well as an implementation of the external functions that are used in the domain.

Step 2: **Definition of the reactors** that will conform the agent. It contains the executive and the deliberative layers of our architecture. It is composed of reactors that share a common interface. Each reactor is responsible of a set of timelines (state variables) of the system (see [11]). In the ERGO architecture, three different types of reactors are envisaged:

- The **Ground control interface reactor** that can be developed by tailoring the Ground control interface services provided in SW1
- The **Mission planner reactor** that can be developed by tailoring the **mission planner reactor** provided in SW1 (containing Stellar, the mission planner).
- A **set of reactors** that interface with the functional layer, that is, forward the commands of the executive and receive observations from the functional layer. These are the so-called “command dispatcher” reactors in the ERGO architecture.

Step 3: Development of the **functional layer** built using a set of TASTE functions (software components), in which some of the components of SW2 can be used.

The ERGO agent is a single TASTE function (i.e. component) that interfaces with the functional layer (a set of TASTE functions) via specific application-interfaces which are also modelled with TASTE. From a TASTE model, code can be generated and deployed for different platforms (namely, Linux and RTEMS).

For the development of the functional layer, the following tasks need to be accomplished.

Step 3.1: Identification of existing software libraries and components that conform to the functional layer. For the majority of the robotic platforms there are existing software modules already developed and tested. The experience accumulated by GMV in the SARGON project [18] showed that the best approach it to take advantage of this fact, reusing this software, generating libraries that can be easily plugged in into TASTE functions.

In some areas, there may not be software modules readily available, but there are existing algorithms that can be wrapped within TASTE functions, or there can be procedures comprising low-level tasks which could be reused.

Step 3.2: Identification of existing TASTE modules that can be used straight away. Any previous components developed in TASTE are ideal candidates for a possible reuse without changes.

Step 3.3: After the identification of modules to be reused or to be designed from scratch or by reusing existing procedures or algorithms, the next task is to define the corresponding views in TASTE: interface view, deployment view, data view and concurrency view.

The constraints to be enforced at run time must be identified. These may come from mission requirements or from constraints on the robotic system. They are required to guarantee the safety of the mission.

The suitability of the processor budget must be assessed. It shall include an estimation of the required resources, mainly CPU and memory.

The interfaces must be made explicit, including the available provided and requested interfaces. This is not only a task of the analysis of the functional layer; the requirements for this will also come from the executive layer.

From the TASTE generated code, the provided interfaces will need to be implemented.

3.2 ERGO Use Cases

Two different use cases have been developed in ERGO.

The first use case is the planetary exploration rover, inspired from the Mars Sample Return (MSR) mission that covers the concepts and requirements of the Martian Long Range Autonomous Scientist, a Martian rover that can be commanded to operate in a multi-sol operations scheme and is able to perform long traverses. For this scenario the SherpaTT rover from DFKI was selected.

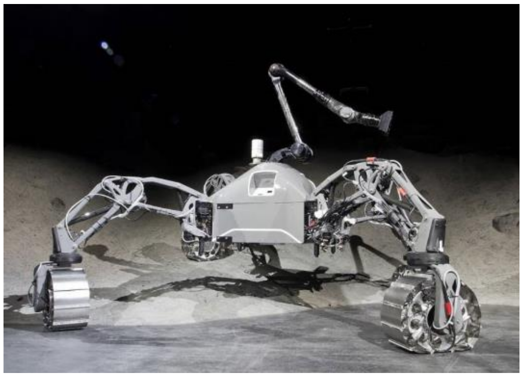

Figure 3: The Sherpa TT robot (courtesy DFKI)

The second scenario chosen is the On-Orbit Servicing mission, where a damaged spacecraft can have one of its modules replaced autonomously by a servicer spacecraft.

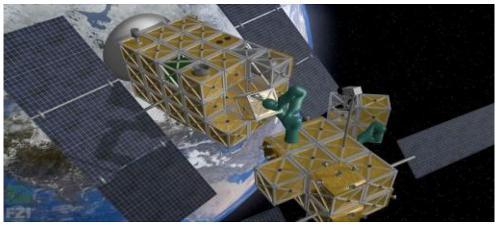

Figure 4: Orbital use case

In the following sections, we describe the use of the framework to build both use cases.

3.3 Planetary Demonstrator

This section describes the ERGO framework instantiation for the Planetary Exploration Demonstrator Software, illustrated in Fig. 5

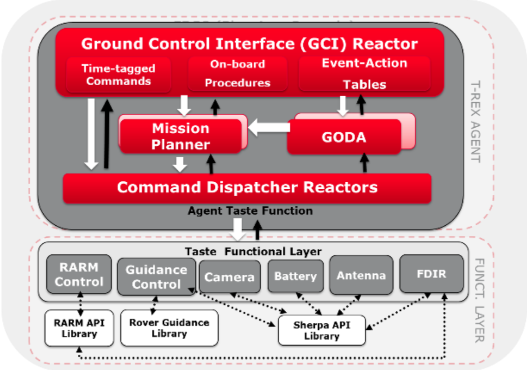

Figure 5: ERGO framework – planetary scenario

Following the methodology described in the previous section, the architecture for the planetary rover consisted on the following components:

- As part of the agent, a Ground Control interface reactor, tailored from the generic component in the ERGO framework, handles use-case specific telemetry and telecommands. It is able to process direct telecommands (E1), time-tagged commands (E2), event-driven actions (E3) and goal commanding (E4), via the mission planner.
- High-level commands (E4) are processed by a dedicated Mission Planner reactor. This component receives high level commands that are used to generate a mission plan. This mission plan, as generated by the planner, contains a set of sub-goals to be executed at given times, together with a set of constraints to be matched. The mission planner reactor uses a specific PDDL domain and problem.
- The agent has also a scientific detector reactor. The so-called **GODA reactor** (that uses the GODA component provided by SCISYS) receives high-level goals in order to detect serendipitous events when the rover is traversing through specific areas by analysing images provided by the camera. When this occurs, a new goal is sent to the mission planner in order to go to a position and take images of the event detected. The planner is then able to

re-plan based on the new goals.

- An additional set of reactors of the agent conform the so-called “Command Dispatcher reactors”: they interface directly to the functional layer. These receive low level goals from the mission planner (e.g. going to a desired position) and receive observations from the functional layer that indicate the results of the execution.

Finally the functional layer consists of a set of TASTE functions developed specifically for this use case, these are:

- **Antenna**: a simulated component for the antenna used to communicate with Ground.
- **Battery**: provides the battery level of the rover.
- **Camera**: interface to the Rover Cameras.
- **FDIR**: A component aimed to detect & isolate errors that could jeopardize the mission during execution.
- **Guidance Control:** Contains the Rover Guidance functionality embedded as a TASTE function. This Function uses the Guidance component detailed in SW2.
- **Planetary Robotic Arm control**: contains the Robotic arm control for the Planetary. This function uses the Robotic Arm component provided in SW2.

3.4 Orbital Demonstrator

The planetary orbital demonstrator is a scenario in which a chaser spacecraft approaches a target, and is able to reconfigure it via a robotic arm. The chaser has a tray that is used to exchange a set of APMs. The chaser will reconfigure the target S/C, so the target must simulate a modular S/C with some faulty/damaged modules which the chaser will have to replace in orbit.

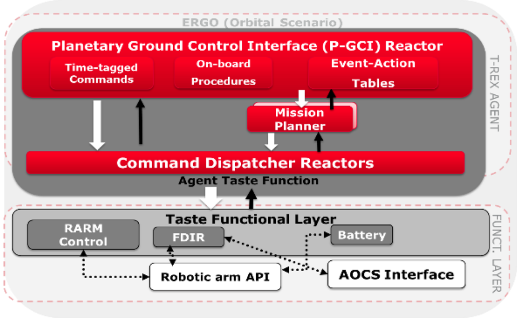

Figure 6: ERGO orbital components

The architecture is depicted in Fig. 6. Note that in the orbital component, the AOCS system is not part of the ERGO components.

The Ground control interface processes E1, E2, E3 and E4 telecommands. High level goals (E4) are sent from Ground to perform changes of the configuration of the target spacecraft. The mission planner identifies the set of operations (elementary operations such as picking, or dropping an APM) to be performed by the robotic arm to reach the desired configuration(s). It does so by using specific PDDL models (domain and problem) generated specifically for this use case.

A specific set of command dispatcher reactors (battery and orbital robotic arm) interface with the functional layer to perform the corresponding commands and received the corresponding observations.

The functional layer developed for this use case includes an FDIR component, a Battery component and a robotic arm component, able to perform robotic arm path planning and execution.

4 V&V APPROACH

Robotic applications have demanding RAMS requirements, which need to be checked for satisfaction on the system design before deployment. This requires the use of formal models and methods, and in the frame of ERGO we propose and extend the BIP framework.

BIP (Behaviour, Interaction, Priority) [14] is a formal language and framework that allows for rigorous component-based system design. The language is based on the well-established theory of Timed Petri Nets/Timed Automata [19], where components are basically modelled as state machines and communicate through (different types of) message exchanges. The BIP tools, described hereafter, provide different analysis techniques at different layers of design granularity which consolidate the confidence on the system’s correctness. We mention that the BIP framework has already been successfully applied for the design and analysis of different robotics systems [20],[21].

The design workflow with the BIP tools is illustrated in Fig. 7. These tools can be directly applied on the ERGO TASTE designs (and not only) via an automated model transformation from TASTE to BIP. This transformation, under development in another PERASPERA SRC project, generates an application software BIP model which can be subject to compositional invariant-based verification with iFinder or automated synthesis of FDIR components with the BIP FDIR tool. Additionally, the design process allows one to refine this model into a distributed system model by integrating the hardware architecture as well as performance constraints. The BIP compiler and engines and SMC-BIP allow simulating and validating the performance constraints of both the application and distributed system model.

As commented in Section 2.1, the ERGO framework includes the iFinder and BIP FDIR tools for formal verification & validation purposes. iFinder checks the satisfaction of safety requirements in a compositional approach: it computes invariants (i.e., constraints that hold at every execution step) for each component independently and next for their interaction based on the system structure. The formula obtained in conjunction with the negation of the safety requirement is given as input to the Z3 SMT solver [23]. If the requirement does not check, the solver returns a counterexample and either the system design must be refined or assumptions are added to the model in the form of invariants. The process is repeated until the counterexample obtained is either a valid one (and then the system is incorrect), or no counterexamples are generated anymore.

The BIP FDIR tool applies automated synthesis techniques to generate from a timed system design (including FDIR architecture), recovery strategies and safety requirements to enforce an FDIR (Fault Detection, Isolation and Recovery) component. We describe in the following the workflow of the tool, for the algorithms devised and their implementation the reader is referred to [22].

In order to apply this tool, several manual steps are needed beforehand. A developer needs to analyse the system requirements in order to obtain an extended system design. Such a design includes the nominal model which should satisfy by default the safety requirements, and the error model which describes what faults are possible in the system and the behaviour after a fault occurs. These models could be obtained separately and incorporated into the extended model through automated automata merging algorithms. Also, one needs to analyse these requirements and connect them to parts (components or sub-systems) of the system design in order to build the FDIR architecture: one or multiple FDIR components which can be centralized or distributed across the platform, monolithical or hierarchical in their structure. Then, with respect to the extended model and FDIR architecture, the recovery strategies are described for each fault type: the functional steps to be executed and their order to bring the system back to the nominal behaviour where requirements hold.

The faults (types) of the extended model are checked for diagnosability. Diagnosability is the fault detection condition provided partial observation of the system: given a set of sensors (possibly minimal) can the occurrence of the fault be identified? Formally speaking, diagnosability checks that given a set of observable actions, there are no nominal and faulty executions that give the same observation (trace). The minimal set of observations as well as the faults most probable to occur for which this condition should be checked can be obtained through model-based safety assessment techniques.

Once this condition is satisfied for all or the most relevant faults, the tool proceeds with synthesizing a diagnoser. The diagnoser is a component that runs in parallel with the system and gives a verdict about whether a fault has occurred or not yet. The diagnoser raises an alarm when the fault is detected which triggers the controller. The controller is based on the recovery strategy specified for the fault (type) and aims to bring the system back to states/modes where the safety requirements hold. The controller is also automatically synthesized by the FDIR tool which assembles it together with the diagnoser(s). Finally, the tool generates C++ code from the synthesized FDIR component that can be put in the original TASTE system design and run online in the actual system. For more details the reader is referred to [22].

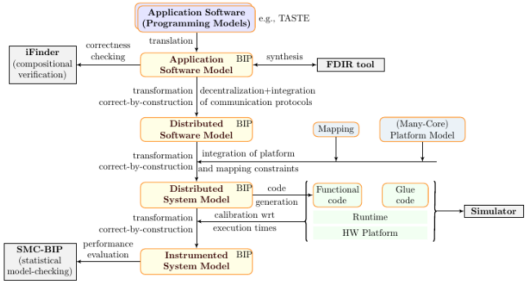
Application Software (Programming Models)
e.g., TASTE
translation
correctness checking
iFinder (compositional verification)
Application Software Model
BIP
synthesis
FDIR tool
transformation correct-by-construction
decentralization+integration of communication protocols
Distributed Software Model
BIP
Mapping
(Many-Core) Platform Model
transformation correct-by-construction
integration of platform and mapping constraints
Distributed System Model
BIP
code generation
Functional code
Glue code
Runtime
HW Platform
Simulator
transformation correct-by-construction
calibration wrt execution times
performance evaluation
SMC-BIP (statistical model-checking)
Instrumented System Model
BIP

Figure 7: System design process in the context of the BIP framework.

5 CONCLUSIONS

The ERGO framework provides a set of building blocks that can be used for the development of any robotic asset requiring a high level of autonomy.

In the ERGO architecture, meanwhile, autonomy levels E1 (telecommanding) to E2 (time-tag commanding) are covered by a (generic) Ground control interface component, E3 and E4 are covered by a mission planning component, which contains an on-board planner and scheduler provided as part of the core framework. An interface with the Functional Layer (the Command Dispatcher reactors) provides the executive layer.

The executive and deliberative layers are embedded into a single agent in charge of the management of a set of control loops. Components of the agent are called reactors and benefit from a single interface.to the agent’s controller. The planner, the Ground Control interface and the agent are designed to be tailored for any specific mission. PDDL models of the system need to be defined, as well as a set of state variables (timelines) handled internally by the agent.

The framework is structured in such a way that it provides core utilities (able to be used in any robotic platform) and specific utilities (covering dedicated functionalities, like the guidance of a planetary rover, the handling of a robotic arm, or the automatic identification of serendipitous events based on images). These components are provided in the form of generic libraries that can be re-used in different developments.

ERGO uses a Model-driven Development (MDD) approach based on TASTE. TASTE is the tool that allows to develop the components of the system, and to define the interfaces among all them. Moreover, the BIP verification and validation tools provided by ERGO can be used to check the correctness of the design.

From the user's perspective, the ERGO framework provides many features, most of them could not be required for a particular robotic asset, but the modularity of the system allows the user to pick those features that are necessary for a particular robotic instantiation.

Acknowledgement

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. In addition, we would like to thank our partners, Airbus Defence and Space Ltd, The University of Basel, King's College University, and Ellidiss, for their support and collaboration in the development of this paper. The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730086.

References

[1] http://www.h2020-ergo.eu/
[2] http://www.h2020-peraspera.eu/
[3] http://exploration.esa.int/mars/
[4] https://mars.nasa.gov/mars2020/
[5] JPL. Mars Exploration Rovers. Available at: http://marsrovers.jpl.nasa.gov/home/index.html.
[6] E. Bernard, Douglas and Gamble, Jr., Edward B. "Remote Agent Experiment DS1 Technology Validation Report". Jet propulsion Laboratory. CIT, Pasadena, California.
[7] G. Rabideau, D.Tran, et al "Mission Operations of Earth Observing One with on-board autonomy" IEEE International Conference on Space Mission Challenges for Information Technology. Pasadena, CA. July 2006.
[8] Medina, A., et al. in "Aerospace Robotics II". Ottawa, Canada: "Online of an Autonomy framework for space robotics". Springer International Publishing pp 187-198, Switzerland 2015.
[9] Ocón Alonso, J. Delfa, J.M. , De la Rosa Turbides, T. García-Olaya, A. Escudero Martín, Y. "GOTCHA: An autonomous controller for the space domain".
[10] Muscettola N., G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt, "IDEA: Planning at the core of autonomous reactive agents", in Proc IWPSS, Houston.
[11] Rajan, K., et al, "A systematic agent framework for situated autonomous systems" AAMAS '10 Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 2, Pages 583-590".
[12] J. Delange and M. Perrotin "On integration of open-source tools for system validation, example with the TASTE tool-chain" 13th Real-Time Linux Workshop.
[13] http://www.h2020-esrocos.eu/
[14] Basu, A., Bozga, M., & Sifakis, J. Modeling heterogeneous real-time components in BIP. SEFM, pp. 3-12. 2016. Fourth IEEE International Conference in Software Engineering and Formal Methods.
[15] ECSS Secretariat. (ESA/ESTEC), "ECSS-E-70-11 Space Segment Operability" (August, 2005). Nordjwick, The Netherlands.
[16] M., Wallace I. and Woods. MASTER: A Mobile Autonomous Scientist for Terrestrial and Extra-Terrestrial Research. s.l.: 13th Symposium on Advanced Space Technologies in Robotics and *Introduction to Terrain-Vehicle Systems*. Ann Arbor, IL: University of Michigan Press. Automation, ASTRA, 2015. 3.
[17] **M. Winter et al** *ExoMars Rover Vehicle: Detailed Description of the GNC System*. s.l. : Proceedings of Space Technologies in Robotics and Automation (ASTRA), 2015.
[18] http://www.sargon-project.eu/
[19] R. Alur and D. L. Dill. A theory of timed automata. Theor. Comput. Sci., 126(2):183–235, 1994.
[20] S. Bensalem, F. Ingrand and 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.
[21] A. Basu, M. Gallien, C. Lesire, T. Nguyen, S. Bensalem, F. Ingrand and J. Sifakis. Incremental Component-Based Construction and Verification of a Robotic System. In European Conference on Artificial Intelligence ECAI'08 Proceedings, volume 178 of FAIA, pages 631-635, IOS Press.
[22] I. Dragomir, S. Iosti, M. Bozga, S. Bensalem. Designing Systems with Detection and Reconfiguration Capabilities: A Formal Approach. Submitted. Extended version available on Arxiv.
[23] L.de Moura, N. Bjørner: Z3: An Efficient SMT Solver, TACAS 2008.
[24] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL-the planning domain definition language. 1998.
[25] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner and Bernhard Nebel. Semantic Attachments for Domain-Independent Planning Systems. In Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009), pp. 114-121. 2009.
[26] Florian Pommerening, Malte Helmert, Gabriele Röger and Jendrik Seipp. "From Non-Negative to General Operator Cost Partitioning". Proc. Of 29th AAAI Conference on Artificial Intelligence (AAAI 2015).