ON-BOARD AUTONOMY VIA SYMBOLIC MODEL-BASED REASONING*

M. Bozzano[1] A. Cimatti[1] A. Guiotto[2] A. Martelli[2]
M. Roveri[1] A. Tchaltsev[1] Y. Yushtein[3]

[1]Fondazione Bruno Kessler
{bozzano,cimatti}@fbk.eu
{roveri,tchaltsev}@fbk.eu

[2]Thales Alenia Space Italy
andrea.martelli@thalesaleniaspace.com
andrea.guiotto@thalesaleniaspace.com

[3]European Space Agency
Yuri.Yushtein@esa.int

Abstract

Deep space and remote planetary exploration missions are characterized by severely constrained communication links and often require intervention from Ground to overcome the difficulties encountered during the mission. An adequate Ground control could be compromised due to communication delays and required Ground decision-making time, endangering the system, although safing procedures are strictly adhered to. To meet the needs of future missions and increase their scientific return, space systems will require an increased level of autonomy on-board. We propose a solution to on-board autonomy relying on model-based reasoning. Our approach integrates many important functionalities (such as plan generation, plan execution and monitoring, fault detection identification and recovery, and run-time diagnosis) in a uniform formal framework. The spacecraft is equipped with an Autonomous Reasoning Engine (ARE) structured according to a generic three-layer hybrid autonomy architecture: *Deliberative*, *Executive* and *Control Layers*. The ARE uses a symbolic representation of the controlled platform. Reasoning capabilities are seen as symbolic manipulation of such formal model. We have developed a prototype of the ARE, and we have evaluated it on two case studies inspired by real-world ongoing projects: a planetary rover and an orbiting spacecraft. For each case study, we have used a simulator to characterize the approach in terms of reliability, availability and performances.

1 INTRODUCTION

Deep space and remote planetary exploration missions are characterized by severely constrained communication links. Limited spacecraft visibility, reduced data rates and high communication latency do not allow for the real-time control by Ground operators. For the surface missions, high level of interaction with the environment may require significant efforts from Ground control, implying high cost of operations. Furthermore, adequate Ground control could be compromised due to communication delays and required Ground decision-making time, endangering the system, although safing procedures are strictly adhered to.

To meet the needs of future missions and increase their scientific return, space systems will require an increased level of intelligence on-board. Taking autonomous decisions through creating their own plans based on up-to-date information, and re-planning in response to unexpected events or anomalous conditions, would greatly improve the efficiency of a mission, system safety, and potentially reduce the cost of Ground operations.

We propose a solution to on-board autonomy relying on model-based reasoning. Our approach integrates many important functionalities (such as plan generation, plan execution and monitoring, fault detection identification and recovery, and run-time diagnosis) in a uniform framework.

The spacecraft is equipped with an Autonomous Reasoning Engine (ARE). The ARE is structured according to a generic three-layer hybrid autonomy architecture: *Deliberative*, *Executive* and *Control Layers*. The *Deliberative Layer* provides goal-driven planning and scheduling, plan validation and system-level fault detection, identification and recovery (FDIR) facilities. The *Executive (Execution Control) Layer* provides facilities to execute and monitor the correct execution of the current mission plan. The *Control (Functional) Layer* provides low-level interactions with the controlled system (sensor acquisition and commands to actuators sending).

The ARE relies on symbolic logic in the Deliberative and Executive Layers, where reasoning is performed on a formal model of the controlled platform. The feedback control loop algorithms of the Control Layer are not based on symbolic reasoning, since complex numerical computations may be involved. However, such computations are directly connected to the formal model through abstraction of the computation results into logical predicates. In this way, the computation steps

*This work is sponsored by the European Space Agency under ESA/ESTEC ITT AO/1-5184/06/NL/JD - ON BOARD MODEL CHECKING.

are interleaved with logical reasoning at the higher levels. The formal model the ARE operates on captures the intrinsic partial observability of the controlled system (available system sensors may not allow for conclusive determination of the controlled components' status). The model is used in the Deliberative Layer for mission plan validation, for re-planning, and for system-level FDIR (e.g. by re-planning to solve the identified problem). The Executive Layer uses the formal model for plan execution and monitoring to detect if an anomaly (i.e. fault, anomalous resource consumption) preventing the achievement of the mission goal occurred. The Control layer uses the model to encode low-level sensor information, and to decode commands to be sent to actuators.

The plans the ARE generates, executes and monitors guarantee the achievement of the mission goal, and facilitate timely reaction to anomalies.

We have developed a prototype of the ARE. It relies on NuSMV, a symbolic model checker for the efficient manipulation of a symbolic representation of a system. On top of its primitives, we have built all the algorithms of the ARE (including plan generation, validation, execution and monitoring, and FDIR). The ARE is largely independent of the controlled system: the upper Layers are application independent, bound to the application domain through the system model description; the dependencies related to the low-level platform interactions are localized in the Control Layer that can be customized through dedicated APIs. The ARE relies on the POSIX C libraries of the RTEMS operating system, and can thus be easily adapted to any system providing POSIX compliant interfaces.

The approach is evaluated on two case studies (a planetary rover and an orbiting spacecraft), both inspired by real-world, ongoing projects. For each case study, a simulator is used to characterize the approach in terms of reliability, availability and performances. The simulators are composed of a functional model of the planetary rover or of the orbiting spacecraft, the on-board software framework and the real hardware target emulator. This architecture allows to evaluate ARE in a platform that is very similar to the one used in real missions.

The evaluation relies on scenarios where re-planning is necessary to take into account possible failures or anomalies caused by changes in the environment, including cases of partial observability resulting from the interaction of failures and anomalies.

This paper is structured as follows. In Section 2 we provide a description of the architecture of the proposed solution. In Section 3 we provide an overview of the formal model we rely upon. In Section 4 we provide an overview of the implementation and of the experimental characterization of the proposed approach on the two case studies. In Section 5 we discuss the proposed approach and we analyze the related work. Finally, in Section 6 we draw some conclusions and discuss future work.

2 THE AUTONOMOUS REASONING ENGINE

Our solution to on-board autonomy consists in equipping the plant to be controlled with what we called an Autonomous Reasoning Engine (ARE). The ARE autonomy is achieved via model based reasoning. This solution allows for the integration of several functionalities in a unique framework: *plan generation*, *plan validation*, *plan execution and monitoring*, and *fault detection identification and recovery*. The ARE interacts with the physical plant and with the Ground Control Station to provide autonomous reasoning capabilities to the plant it aims to control. From the physical plant, it receives information from the sensors, and delivers control commands to the actuators. With the Ground Control it will exchange information on the mission goals and initial mission plan, and it will also receive direct commands to activate ARE functionalities and to inspect the status of the ARE.

In order to perform the reasoning, the ARE uses a formal model describing the behavior of the plant it aims to control and the environment in which it will work. The formal model describes both the nominal and the degraded or faulty behavior of the controlled plant. The formal model takes into account that the status of the controlled plant and of the environment is not completely observable. Moreover, to facilitate the deliberative actions it separates out the discrete control part and the continuous parts (resources like power consumption, acquired data, etc.). The ARE uses assumptions that are expected to always hold during the executions. These assumptions represent constraints on the faults that may occur, on the environment (e.g. the external temperature is below a certain threshold), and on the resources (e.g. the energy of the batteries is above a certain threshold). The formal model used by the ARE contains fault bits in order to have an observation on the occurred faults. These fault bits are assumed to provide a perfect sensing. Each fault bit is also associated with a context dependent probability to be used by fault identification in the case there are multiple candidate faults, thus deciding for the fault with highest probability. The mission plans MP manipulated by the ARE are contingent plans that aim on one side to guarantee the achievement of the mission goal despite the non determinism of the environment and the partial knowledge of the controlled plant at run-time. On the other side, they aim at facilitating the monitoring of the plan itself to detect if some anomaly (i.e. assumptions no longer satisfied or anomalous resource consumption) occurs as to react as soon as possible. In order to facilitate plan monitoring, the plans are labeled with the

assumptions that are expected to hold during plan execution.

From the logical point of view, the ARE is structured according to a generic three layers hybrid autonomy architecture: Deliberative layer, Executive layer, and Control layer (See Figure 1).

The *Deliberative Layer* (also called Decision Layer) of the ARE is the top layer. It is responsible for generating mission plans, for validating the generated plans to ensure that they are guaranteed to achieve the mission goal, and for triggering re-planning whenever needed (e.g., when the plan is no longer guaranteed to achieve the goal due to a run-time anomaly or a change in the operating environment). The Deliberative layer is responsible for sending to the Executive layer valid plans to start executing them. The Deliberative layer also includes an FDIR block which is activated in response to an anomaly (e.g., a fault or an assumption violation) detected by the Deliberative layer itself or by a lower layer to identify and then recover from the anomaly. In order to recover from the anomaly, the FDIR block can trigger the most appropriate functionalities, depending on the nature of the anomaly itself, computing the new assumptions, and triggering plan validation or re-planning, if needed. Several recovery strategies can be embedded in the Deliberative layer depending on the degree of autonomy that we would like to achieve, on the complexity of the controlled plant, and on the criticality of the identified problem. If the anomaly is due to a change in the environment or in the expected use of resources, then new assumptions are computed and different actions can be performed. For instance, it is possible to validate the rest of the plan w.r.t. the new assumptions. If the validation of the plan succeeds then the execution of the plan can continue. Otherwise, a re-planning activity with the new assumptions can be triggered. At the end of the FDIR, in order to decide whether to continue the execution of the remaining part of the plan, or to attempt a re-planning or moving into a safe mode state, the ARE first validates the remaining plan to execute. If the plan is valid, then the execution of the remaining part of the plan continues. Otherwise, it tries to re-plan from the current configuration. If the plan generation succeeds then the new computed plan is executed. Otherwise, the ARE moves to a safe state mode waiting for intervention from Ground.

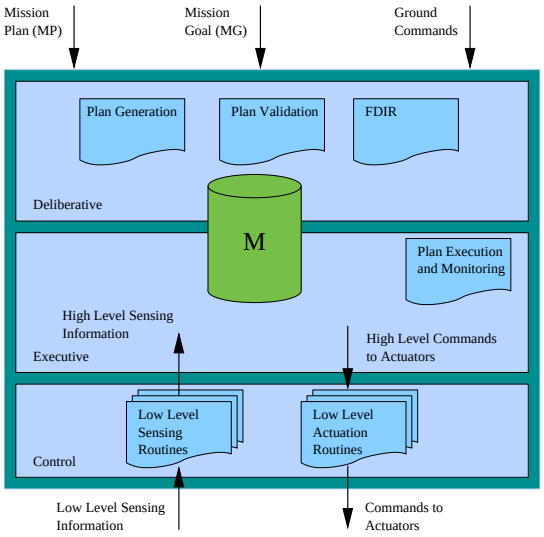

Fig. 1. The ARE architecture.

The *Executive layer* is responsible for executing a given mission plan coming from Ground or generated by the upper layer. It is also responsibility of the executive layer to carry out plan monitoring and, in case an anomaly (e.g., an assumption that is no longer satisfied, or an anomalous resource consumption) to detect this in order to give the control back to the Deliberative layer, that reacts by activating the FDIR block. The Executive layer reads information on the status of the controlled plant; depending on this status it takes the proper branch of the contingent plan it aims to execute; it checks whether the current command to execute according to the mission plan can be performed, and whether the assumptions (labeling the plan to be executed) expected to hold for the whole execution of the plan still hold. In the Executive layer, there is no explicit distinction between assumptions that do not hold because a controlled component that failed, and those that do not hold because of a change in the external environment. The Executive layer collects diagnostic information (performed observations, executed commands, assumption violations) in order to provide them to the FDIR block in the Deliberative layer, whenever requested, to identify the possible cause of the anomaly and activate the proper recovery function.

The *Control layer* implements the conversion between the model-based level and the lower level of the physical plant. It contains the low-level code that is responsible for the acquisition of sensing information and for sending low level commands to the actuators. The control layer is implemented as a set of software procedures that are tailored to the control and monitoring of specific physical devices. This layer also provides to the higher layers functions to estimate the resource consumption resulting from the execution of a command. This information is then used by the Deliberative layer to validate the generated or received from Ground mission plan w.r.t. resource consumption, and by the Executive layer to monitor the correct execution of the current mission plan.

The ARE functionalities are triggered according to a predefined Finite State Machine (FSM) that activates the different building block functionalities provided by the ARE to achieve the desired degree of autonomy. This FSM is responsible for responding to possible triggers coming from Ground or triggers coming from lower layers (e.g. because of a problem detected during plan execution) to activate FDIR functionalities, to re-plan or re-validate the remaining plan.

3 MODELING AND REASONING

The formal framework the ARE relies on for the modeling and reasoning builds upon and extends the one used in the novel and emerging framework known as *Planning as Symbolic Model Checking* [9, 13, 10, 4, 5, 2, 3, 11, 12]. The works in [9, 13, 10] tackle the problem of finding a plan that will guarantee the achievement of a goal despite the non-determinism in the initial condition and in action effects under the assumptions that the controlled plant is fully observable. The subsequent works extended the previous works by considering also partial observability [4, 5, 2, 3] and the extreme case of no observability (conformant planning) [11, 12]. This general framework allows for a harmonic integration of the deliberative, executive and monitoring functionalities the ARE provides in a unique well founded formal framework. A precise description of the formal model can be found in [6].

The formal model used by the ARE describes the behavior of the plant it aims to control and the behavior of the environment in which the plant itself will work. Such model allows to capture the possible non-determinism in commands effects and in the environment. Moreover, it also allows to model the fact that the status of the controlled plant and of the environment can be partially observable due to the limited availability of sensors. The formal model specifies also the behaviors the controlled plant can end up during the normal execution because of possible faults. The faults, as any part of the controlled plant, can be partially observable. To each faults we associate a probability that tells which is the probability for that fault to occur. This probability is then used by FDIR to decide for the fault with highest probability in the case multiple candidate faults are identified. The formal model adopted in this approach separates out the discrete control part and the continuous parts (e.g., resources such as power and data) of the controlled plant to facilitate the deliberative actions. The link from the continuous and the discrete control parts is modeled in the formal framework by means of resource estimation functions that will allow to estimate the value of the continuous controlled variables and to estimate the variation of those variables due to a command execution. These functions corresponds to the estimation functions the Control layer provides to the higher layers.

Because of partial observability when we read the sensors we cannot know precisely the state in which the plant really is. Moreover, because of the non-determinism in action effects it is also not possible to know precisely in which state the controlled plant end-up after the execution of a command. To formally capture uncertainty in the framework of planning under partial observability [5, 3], the concept of *belief state* is introduced. Intuitively, a *belief state* is a non empty set of states representing the uncertainty in which the controlled plant currently is.

In this work we consider *reachability* goals and *strong contingent* plans [5, 3]. A reachability goal is represented by set of states the plant should achieve. Strong contingent plans at execution time sense the world and, depending on the state of the world, can execute different actions, and are guaranteed to achieve the goal despite the non-determinism and partial observability of the model. A strong contingent plan is either: an empty plan ϵ; a sequence $a :: \mathcal{P}$, where a is an action; a conditional plan $ite(o, \mathcal{P}1, \mathcal{P}2)$, where o is an observation. An empty plan ϵ denotes the end of the plan. A plan $a :: \mathcal{P}$ indicates that it is necessary to perform action a and then plan $\mathcal{P}$. A conditional plan $ite(o, \mathcal{P}1, \mathcal{P}2)$ indicates that depending on the value of observation variable o it is necessary to execute either $\mathcal{P}1$ (if o is true) or $\mathcal{P}2$ (if o is false). The purpose of checking the value of an observation variable is to partition the current belief states into smaller and more precise subsets and apply consecutive plans depending on the states the system is in. The execution of a contingent plan from a belief state is performed by computing the set of states that can be reached by applying the contingent plan and checking that the action can be applied in the current belief state. Intuitively, a contingent plan is a strong solution to the planning problem of reaching a goal, if the belief state resulting from the execution is included in the goal.

To formalize constraints which the controlled plant is supposed to obey at execution time and expectations that the ARE has on the execution environment, we use assumptions. Such assumptions are used to simplify plan generation by reducing the search state space, and in plan validation and monitoring to ensure that during the plan execution such assumptions are satisfied. For this work we restrict to assumptions of invariant type, i.e. that are expected to hold at any time point of the execution of the plan.

To allow for the monitoring of the satisfaction of the assumptions during the execution of the plan, each step of the contingent plan is annotated with a pair of sets of states $\langle S_g, S_{pg} \rangle$. S_g consists of the states where the assumptions hold (good states). Set S_{pg} includes S_g and may additionally include states not satisfying the assumption that are indistinguishable from S_g (possibly good states). If the assumptions always hold throughout the execution of a given plan, then at each step the current belief state is a subset of S_g. If, on the other hand, the current belief state is not included in the set S_g, but it is included in the set S_{pg}, then the assumptions may have been violated but the violation cannot be detected because of partial observability. If the current belief state is not included in S_{pg}, then the assumptions have been violated.

Plan validation is performed by checking that the plan adheres to the structure of an annotated contingent plan and that goal achievement is guaranteed despite the non-determinism and the partial observability. This is achieved by simulating the plan. During the simulation, additional checks are performed to verify that there will be enough resources to execute the plan to completion. Specifically, we check whether the estimated resources at each point of the plan execution are such

that they will allow the current action to be executed. If this is not the case, plan validation fails. During the validation of the plan, the estimation resource consumption functions provided by the Control layer are used to check whether the resources do not go below minimum values, and thus will allow for the achievement of the goal.

The algorithm for computing such annotated strong contingent plans is a simplified version (dealing only with invariants) of the algorithm presented in [3]. The algorithm performs a forward AND/OR search in the space of belief states starting from the initial belief state. AND nodes correspond to observations (*ite* plan nodes), while OR nodes correspond to action execution. In the forward AND/OR search two belief states are progressed. These two belief states correspond to the two belief states S_g and S_{pg} used to monitor the satisfaction of the assumptions during execution, and represent the uncertainty w.r.t. assumptions status. We refer the reader to [3] for additional details.

FDIR is performed by keeping a history window of the performed observations and the actions executed during plan execution. Whenever a failure is detected while monitoring the execution of the plan, a monitor for the evolution of the fault variables is built and composed with the model. Then, on the resulting model the observation window is re-simulated by accumulating the set of states. The final set of states is then analyzed to identify the possible faults, if any. If more than one fault is identified in this phase, then the one with higher probability is selected, and the corresponding recovery action is activated.

4 IMPLEMENTATION AND EXPERIMENTAL VALIDATION

The ARE software prototype has been built on top of the NuSMV symbolic model checker [8]. Figure 2 depicts a high level description of the software architecture of the ARE. The NuSMV library provides all the low level routines for the symbolic manipulation of the discrete model of the controlled plant, and for implementing all the building blocks necessary for the implementation of the high level autonomy functionalities provided by the ARE (i.e. plan generation, plan validation, plan execution and monitoring, and for the identification of the possible faults). These functions have been implemented following the "standard" approach at the basis of the Planning as Model Checking framework [9, 10]. The ARE Functions include all the functions to perform plan generation, plan validation, plan execution and monitoring, and to perform FDIR. Moreover, this layer provides auxiliary functions to upload a new model of the controlled plant, to upload new assumptions, to upload the mission goal, to upload a new mission plan and to build the corresponding internal symbolic representation. In this software block are also implemented and invoked the low level routines responsible for the decoding of low level commands to be sent to the actuators, and low level functions to read the sensors and to build the internal representation to be used for the deliberative and executive activities provided by the ARE. The ARE top level provides the implementation of the ARE top level Finite State Machine (FSM) and the mechanism to activate the ARE functions. The ARE top level computation is decomposed in two threads: a master and a child. The master is the thread in which the ARE is initially run. This thread initializes the child thread and implements the FSM that prescribes the behavior of the ARE. The child thread performs all the ARE functionalities associated to the different states of the ARE finite state machine (initialization, plan generation, plan validation, ...). The master gives commands to the child to activate the corresponding functionality, or wait if nothing has to be done. The master may stop the activities currently in execution in the child thread to react to possible problems occurring during the execution (e.g. an anomaly is detected, a command from Ground has been received, etc.). We remark that the whole software is built on top of the POSIX C interface. Thus, the software can be deployed on any operating system providing a POSIX C interface (e.g. under RTEMS, Linux, Solaris, Windows).

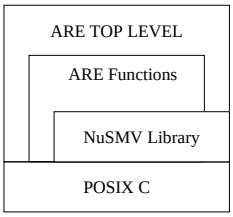

Fig. 2. ARE SW Architecture.

We experimented the proposed approach on two case studies, a planetary rover, and an orbiter. Both case studies are extracted from real-world domains, and interface the ARE with a simulator. Spacecraft simulator is based on a library that contains the basic common functionalities (TC management, TM management, software bus services). The services offered by this library allow to easily integrate software components into a software bus. The mock-up is running on a Linux PC host with a simulator (currently SiS [18] on ERC32 and TSIM [19] on LEON3) or a real target (FPGA board or ASIC board). Commanding (TC, scripts) as well as display and reporting (TM, reports, checks) are network transparent and support multiple users on the same simulation. For each of the case studies, we have identified realistic objectives, and instructed the simulator to present both nominal and anomalous conditions for planning and re-planning. Several simulations have been performed in order to evaluate the suitability of the approach w.r.t. different metrics. For instance, the rover mission characteristics as unpredictable local conditions on the planetary surface and with time-variable conditions; the rover mission constraints as limited bandwidth, intermittent visibility, long round trip delay; rover system operations to perform a measurement cycle that included movement, sample acquisition and sample preparation and distribution; the orbiter mission characteristics; the orbiter mission constraints; the orbiter system operations; and finally the development methodology. The approach has been experimented on two different simulation scenarios: (i) re-planning after fault; (ii) re-planning after an unexpected change in environment. The system has been run in order to characterize the approach on the following parameters: reliability (requirements coverage, generated plan compliance

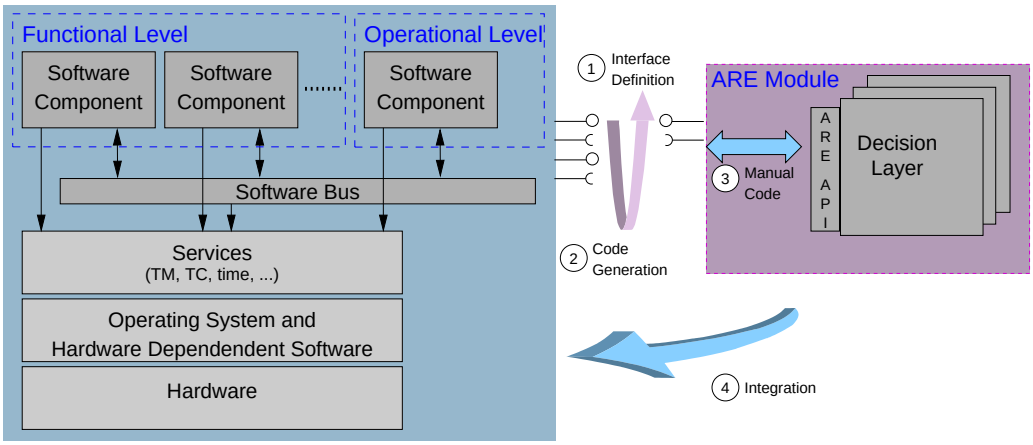

Fig. 3. The ARE interacting with the simulator.

with the goal); availability (reaction time); and performance (processing power and memory required). The system has been characterized both from the functional point of view, and from the performance point of view. A typical scenario used to evaluate the system in terms of performance includes: initialization of models, loading of plan, plan validation, plan execution, anomaly detection and analysis, recovery, and finally plan execution.

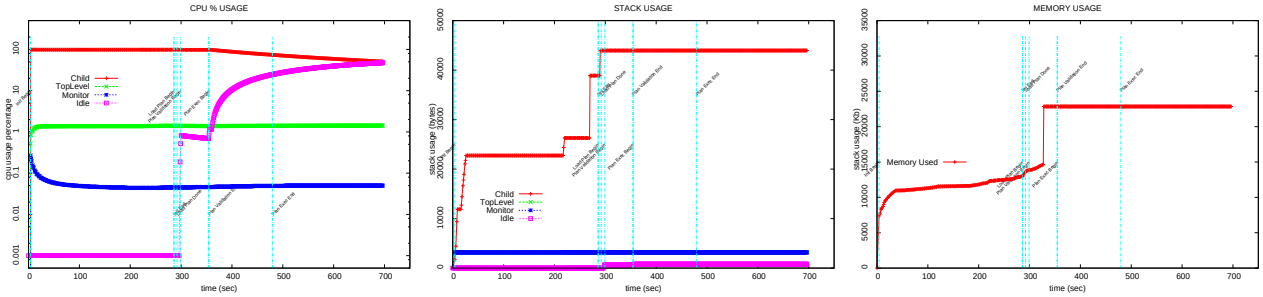

Fig. 4. The ARE characterization on TSIM for LEON3 for the Planetary Rover.

In Figure 4 it is reported an excerpt of the characterization we did on the typical scenario for an observation window of 700 seconds using the primitives provided by RTEMS 4.9 [17]. Four threads are running on the platform: the *Idle* thread, the *Main* threads that implements the ARE FSM, the *Child* thread that is responsible of the different ARE functionalities, and the *Monitor* thread that contains the code to extract performance figures. The first plot on the left reports the percentage of CPU usage for the considered observation window. The plot in the middle reports the maximum stack used over time by the different threads. Finally, the plot on the right reports the memory used by the ARE in the given period. The vertical bars marks the beginning and end of the different steps. From left to right we have: initialization, loading of the plan, validation of the plan, execution of the plan, and waiting for new commands to arrive. The plots show that the CPU intensive steps are the initialization of the model, the loading of the plan and its validation. Once the execution of the plan starts, the child thread decreases its use of CPU since during the execution of a low level command it waits for the command to terminate before going on with the execution of the remaining part of the plan. At the same time the Idle thread increases its percentage use. Even though not reported in the plot, we remark that, an increase of CPU usage of the Child thread and a decrease of Idle CPU usage will be showed while performing plan generation. The Main thread, instead, is using a very low percentage of CPU (about 1.5%). The Monitor on the other hand uses a practically constant amount of CPU which is about 0.05%. The plots also show that the memory used by ARE grows during the initialization and also during the plan loading and validation. During plan execution it is almost constant, and it remains constant after the execution of the plan terminates. The reason for this behavior resides in the NuSMV internals used to represent the model. We refer the reader to [8] for a thorough discussion. We remark that, there is no dynamic memory allocation in the ARE implementation. This analysis has been performed to evaluate the memory requirements for ARE operations.

The ARE software and related material can be downloaded from `http://es.fbk.eu/projects/esa_omc-are` upon request to the authors.

5 DISCUSSION AND RELATED WORK

This approach extends the classical approach of Planning as Model Checking by allowing to consider also the continuous component of the plant. This is achieved by decoupling the logical reasoning and the numerical computation. Typically the autonomy aspects of plan generation, plan validation, plan execution and monitoring, and FDIR, are addressed with different models, usually manually converted, of the controlled plant, and thus prone to problems due to possible differences in the conversion. This framework integrates in a uniform framework all the different aspects for a high degree of autonomy, hence avoiding the problem of conversion among the different models used in the different aspects. The formal model allows for a formal verification with state-of-the art verification techniques nowadays used in everyday activities in different applicative domains (e.g. hardware and software designs). The characterization performed within this project, shows that although the work is in a prototype state, it is suitable to be embedded in current space applications and on-board computers.

The first notable approach to model-based autonomy is the Deep Space One (DS1) experiment (1998-2001) by the NASA agency. The DS1 was equipped with a "Remote Agent"(RA) [1] module that provided model-based executive capable of goal-driven planning and scheduling, diagnosis and recovery from injected faults. The model-based executive of the RA is the Livingstone model [20]. Titan [15] (developed by MIT) is the descendant of the Livingstone model. This executive is composed of two main function: a mode estimation and a mode reconfiguration. Mode estimation is responsible for updating the current state by taking into account the commands that have been issued, and the observations perceived from the controlled plant. This is performed by taking into account the most likely possible state that is compatible with the history of executed actions, with gathered observations, and with the plant model. Mode reconfiguration is responsible for updating the current status, making sure that the specified actions are still applicable and valid. The RA is similar in spirit to the one proposed in this work. Our approach differentiates from the RA since the same formal model is used in all the phases from the deliberative to the executive levels.

MUROCO-II [16] is a support tool for a generic formal framework for the specification and verification of space missions. It implements a three layers framework, but the whole approach is off-line. Basically, actions and tasks of a mission can be specified and validated. The framework relies on the Esterel language, and simple temporal properties can be simulated, and formally proved. Our approach extends MUROCO-II in two main directions. First, MUROCO-II is a framework for an off-line activity taking place on ground, while the current approach focuses on technologies and tools for on-board autonomy. Second, the system developed in MUROCO is unable to deal with diagnosis, since the tools in Esterel focus on verification; planning is also out of reach for all those cases where non-determinism has to be taken into account (with the environment interpreted in an adversarial manner); in our approach, both diagnosis and planning are covered.

The MMOPS [14] approach develops an on-board system, TVCR (Timeline Verification, Control and Repair), that takes into account scheduling issues, and is able to carry out limited amounts of re-planning. The objective is to try and detect whether the mission time line (MTL) currently being executed is still likely to achieve its goals and not to cause trouble given that the current conditions may have departed from the estimated ones, and in case of detected problem suggests possible repairs of the MTL. The main components are a plan validator, an execution monitor, and a plan repair generator. The form of planning is very specific, and does not address the problem of defining a generic automated reasoning system to be reused in different settings and for different functionalities. Our approach implements functionalities similar to the ones of MMOPS, but in a unique formal framework within a three layers hybrid autonomous architecture.

6 CONCLUSIONS AND FUTURE WORK

The developed approach provides the basis for the unified approach to on-board autonomy. It integrates the goal-driven operation of a spacecraft with the fault management capabilities, hence facilitating a high level of autonomous operation in the presence of faults and in the partially unknown dynamic conditions of the operational environment. The developed ARE represents an autonomy building block suited for use in the existing spacecraft system architectures. The ARE is largely independent from the system it aims to control. The Deliberative and Executive layer reasoning is bound to a concrete system through the model of the system. The only required customizations are localized to the Control layer for integration with the system-specific low-level interactions with the controlled system. This approach makes ARE usable in any autonomous system with goal-driven operation and observation-based control. The employed reasoning approaches can cope with non-deterministic operational environment of a spacecraft, which makes the project results highly relevant for the surface interaction environments (planetary rovers), where the environment-induced anomalies are of non-deterministic nature. The ability of the developed autonomy component to provide nominal operation under the

conditions of partial observability of the system and its environment facilitates the ARE use in systems with limited or suboptimal sensor data and sensing abilities.

The developed ARE prototype has been evaluated on the space-rated embedded target (running under RTEMS on the LEON3 processor). The obtained performance data shows ARE usability in the context of the current space applications and available on-board computers.

There are several direction for improvements to this approach, at different levels. For instance, the reasoning engine currently uses BDDs [7]. It would be worth to modify the reasoning engine of the ARE to use techniques based on propositional satisfiability (SAT) provided by the NuSMV library. These complementary techniques allows to deal with problems out of the scope of BDD based approaches (see [8]). As far as goals are considered, it would worth extending the approach to sequence of goals. As far as the assumptions are considered, it would be worth extending the approach to specify more complex assumptions like in [3]. Also the engine, that currently uses the NuSMV library, could be improved by providing a dedicated library with a further reduced code to adapt better to its application in systems with limited computation resources.

REFERENCES

[1] Remote Agent. http://ic.arc.nasa.gov/projects/remote-agent/.

[2] A. Albore and P. Bertoli. Generating safe assumption-based plans for partially observable, nondeterministic domains. In Deborah L. McGuinness and George Ferguson, editors, *AAAI*, pages 495–500. AAAI Press / The MIT Press, 2004.

[3] A. Albore and P. Bertoli. Safe LTL Assumption-Based Planning. In D. Long, S. F. Smith, D. Borrajo, and L. McCluskey, editors, *ICAPS*, pages 193–202. AAAI, 2006.

[4] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In *Proceedings of IJCAI-01*. AAAI Press, August 2001.

[5] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Strong planning under partial observability. *Artif. Intell.*, 170(4-5):337–384, 2006.

[6] M. Bozzano, A. Cimatti, M. Roveri, and A. Tchaltsev. ARE Domain Formal Specification. Technical report, Fondazione Bruno Kessler, 2008. http://es.fbk.eu/projects/esa_omc-are.

[7] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comp.*, 35(8):677–691, 1986.

[8] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A New Symbolic Model Checker. *STTT*, 2(4):410–425, 2000.

[9] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via Model Checking: A Decision Procedure for $\mathcal{AR}$. In *Proceeding of ECP-97*, volume 1348 of *LNAI*, pages 130–142, Toulouse, Fr, Sept. 1997. Springer.

[10] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2):35–84, 2003.

[11] A. Cimatti and M. Roveri. Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research (JAIR)*, 13:305–338, 2000.

[12] A. Cimatti, M. Roveri, and P. Bertoli. Conformant planning via symbolic model checking and heuristic search. *Artif. Intell.*, 159(1-2):127–206, 2004.

[13] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proceeding of AAAI-98*, Madison, Wisconsin, 1998. AAAI-Press.

[14] M. Woods et al. Mars Mission On-Board Planner and Scheduler (MMOPS) Summary Report, Issue 1, ESA Contract 17987/03/NL/SFe CCN1, 2006. ftp://ftp.estec.esa.nl/pub/wm/wme/obmc/MMOPS-SUMRPT.pdf.

[15] L. Fesq, M. Ingham, M. Pekala, J. Van Eepoel, D. Watson, and B. C. Williams. Model-based autonomy for the next generation of robotic spacecraft. http://groups.csail.mit.edu/mers/papers/IAC02_MIT_paper.pdf.

[16] K. Kapellos. Formal Robotic Mission Inspection and Debugging (MUROCO II) Executive Summary, Issue 1, ESA Contract 17987/03/NL/SFe, 2005. ftp://ftp.estec.esa.nl/pub/wm/wme/obmc/MUROCO-TRA-EXSUM.pdf.

[17] RTEMS — Real-Time Operating System for Multiprocessor Systems. http://www.rtems.com.

[18] SPARC instruction simulator (SIS). http://www.esa.int/TEC/Software_engineering_and_standardisation/TECS2BUXBQE_0.html.

[19] TSIM2 LEON3. http://www.gaisler.com.

[20] B. C. Williams and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *AAAI/IAAI, Vol. 2*, pages 971–978, 1996.