A Comprehensive Approach to On-Board Autonomy Verification and Validation *

M. Bozzano, A. Cimatti, M. Roveri, A. Tchaltsev
Fondazione Bruno Kessler
{bozzano,cimatti,roveri,tchaltsev}@fbk.eu

Abstract

Deep space missions are characterized by severely constrained communication links. To meet the needs of future missions and increase their scientific return, future space systems will require an increased level of autonomy on-board. In this work, we propose a comprehensive approach to on-board autonomy relying on model-based reasoning, and encompassing many important reasoning capabilities such as plan generation, validation, execution and monitoring, FDIR, and run-time diagnosis. The controlled platform is represented symbolically, and the reasoning capabilities are seen as symbolic manipulation of such formal model. We have developed a prototype of our framework, implemented within an on-board Autonomous Reasoning Engine. We have evaluated our approach on two case-studies inspired by real-world, ongoing projects, and characterized it in terms of reliability, availability and performance.

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 real-time control by Ground operators. For surface missions, a high level of interaction with the environment may require significant efforts from Ground, 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, future space systems will require an increased level of intelligence on-board. Taking autonomous decisions by creating 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.

*This work was sponsored by the European Space Agency under contract ITT-AO/1-5184/06/NL/JD - On Board Model Checking.

In this paper we propose a comprehensive formal framework for on-board autonomy with on-ground support. The novelty of the proposed solution lies in the definition of a formal framework, comprehensive of heterogeneous functionalities (plan generation, execution and monitoring, fault detection, isolation and recovery, and run-time diagnosis), suited both for on-ground and on-board reasoning. This framework relies on a symbolic representation of the system under control, and allows one to capture its intrinsic partial observability (available system sensors may not allow for conclusive determination of the system status). This framework enables for the formal validation of the model of the controlled system used for the deliberative, execution and verification activities. We propose to use safe assumption-based contingent plans. At execution time, these plans sense the world and, depending on its state, may execute different actions. Moreover, they are annotated with conditions that help monitoring whether the assumptions are satisfied during the execution. Our framework separates the discrete part of the system under control from the continuous parts (e.g., power consumption) to facilitate deliberative reasoning.

This solution has been developed in response to an invitation to tender of the European Space Agency aiming at developing an integrated approach for model based on board autonomy [OMC-ARE, 2008]. We have developed 1) an Autonomous Reasoning Engine (ARE) for spacecraft missions, structured according to a generic three-layers hybrid autonomy architecture that are responsible for deliberation, execution, and low-level control, and 2) a prototype of the ARE, based on the NuSMV model checker [Cimatti *et al.*, 2000]. The ARE is largely independent of the controlled system, and can be easily adapted to any system providing POSIX compliant interfaces. As requested by ESA, we have evaluated our approach on two case-studies inspired by real-world, ongoing projects, and we have characterized it in terms of reliability, availability and performances using a spacecraft simulator, running on-board software on real hardware target emulators currently used in ESA, hence very close to the platforms used in real missions. The successful performance characterization showed practical feasibility of the approach. Quoting the ESA project officer, "initially we would have never thought that you would be able to run on such hardware".

This paper is structured as follows: we first present the modeling and reasoning framework, then we discuss the

ARE, we present the experimental characterization, and finally we discuss related work and draw some conclusions.

2 Modeling and Reasoning Framework

2.1 Formal Model of the System

We model the system under control following the *Planning as Model Checking* approach presented in [Cimatti *et al.*, 2003b; Bertoli *et al.*, 2006]; we extend it to enable reasoning about resources (e.g., battery power, acquired data). Note that the model of the system includes both the nominal behavior and the behavior in presence of faults, and that our formalization is independent of the language used to specify the model.

Definition 1 (System) *A system is a tuple* $\mathcal{M} = \langle \mathcal{Q}, \mathcal{I}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{X}, \mathcal{R}, \mathcal{R}_\mathcal{E} \rangle$ *where:*

- $\mathcal{Q}$ *is a finite set of states;*
- $\mathcal{I} \subseteq \mathcal{Q}$ *is a set of initial states;*
- $\mathcal{A}$ *is a finite set of actions;*
- $\mathcal{T} : \mathcal{Q} \times \mathcal{A} \to 2^\mathcal{Q}$ *is the transition relation;*
- $\mathcal{O}$ *is a finite set of observations;*
- $\mathcal{X} : \mathcal{Q} \to 2^\mathcal{O}$ *is the observation function;*
- $\mathcal{R}$ *is a finite set of resources;*
- $\mathcal{R}_\mathcal{E} : 2^\mathcal{Q} \to (\mathcal{R} \to \mathbb{R}^2)$ *is a resource estimation function.*

We require that $\mathcal{X}(s) \neq \emptyset$ *for every* $s \in \mathcal{Q}$.

This representation separates out the discrete control part and the continuous parts, in order to facilitate model based validation and deliberation. The two parts are associated via the resource estimation function $\mathcal{R}_\mathcal{E}$, that provides an estimate of resources (lower and upper bounds) in a set of states. For $\mathcal{R}_\mathcal{E}(S, r) = \langle m, M \rangle$, with $m \leq M$, we use $\mathcal{R}_\mathcal{E}(S, r)^l = m$ and $\mathcal{R}_\mathcal{E}(S, r)^h = M$ respectively to refer to the lower and upper bounds (represented as rational numbers) of the resources r in S. We say that $\mathcal{R}_\mathcal{E}(S_1) \leq_\mathcal{R} \mathcal{R}_\mathcal{E}(S_2)$ ($\mathcal{R}_\mathcal{E}(S_1)$ is less then, or equal to, $\mathcal{R}_\mathcal{E}(S_2)$) iff $\forall r \in \mathcal{R}.\mathcal{R}_\mathcal{E}(S_1, r)^h \leq \mathcal{R}_\mathcal{E}(S_2, r)^l$, and $\mathcal{R}_\mathcal{E}(S_1) \not\leq_\mathcal{R} \mathcal{R}_\mathcal{E}(S_2)$ for its negation.

Given an action $a \in \mathcal{A}$, the *precondition* of an action a is the set of states $pre(a) = \{s \in \mathcal{Q} \mid \exists s' \in \mathcal{Q}, \langle s, a, s' \rangle \in \mathcal{T}\}$. An action a can be applied to a state $s \in \mathcal{Q}$ (set of states S) only if $s \in pre(a)$ ($S \subseteq pre(a)$). Otherwise, s has (S contains states with) no successors if action a is applied. We write $S[o, \top]$ to denote the set $\{s \in S | o \in \mathcal{X}(s)\}$, of states compatible with observation o, and dually $S[o, \bot]$ to denote the set $\{s \in S | \exists o' \in \mathcal{X}(s), o' \neq o\}$ of states that are compatible with any observation other than o. The set of states *indistinguishable* from a set $S \subseteq \mathcal{Q}$, written $\textsc{Ind}(S)$, is the set $\textsc{Ind}(S) = \{s \in \mathcal{Q} \mid \exists s' \in S. \forall o \in \mathcal{O}(o \in \mathcal{X}(s) \iff o \in \mathcal{X}(s'))\}$. The set of states indistinguishable from S always includes S, i.e., $S \subseteq \textsc{Ind}(S)$.

2.2 Model validation

With a formal model at hand, it is possible to use techniques such as model checking [Clarke *et al.*, 1999] to validate its behavior. Model checking allows one to exhaustively verify the model against a set of properties typically expressed in temporal logic, e.g., PSL [Eisner and Fisman, 2006]. Underlying technologies include BDD-based [Bryant, 1992] or SAT-based [Biere *et al.*, 2003] techniques.

To manage the complexity of the model and of the validation, techniques such as predicate abstraction [S. Graf and H. Saidi, 1997] and Counterexample Guided Abstraction Refinement (CEGAR) [Clarke *et al.*, 2003] can be applied: an abstraction of the concrete system is computed [Lahiri *et al.*, 2006]; a counterexample trace for the abstract system is built; if the trace has a counterpart in the concrete system, then a genuine counterexample has been found; otherwise the abstraction will be refined, and the loop iterated.

2.3 Plan generation, validation, execution and monitoring

The planning problem consists of finding a *plan* whose execution guarantees the achievement of the given goal. In the planning community, different notions of goal and different solutions have been studied, see e.g., [Ghallab *et al.*, 2005]. Here, we restrict ourselves to consider reachability goals [Cimatti *et al.*, 2003b; Bertoli *et al.*, 2006], that are characterized by a non-empty set of goal states $\mathcal{G} \subseteq \mathcal{Q}$ that the system under control is aimed to achieve.

In this work we consider *weak* and *strong* plans [Cimatti *et al.*, 2003b]. Weak plans are plans that have a chance to reach the goal, while strong plans are plans that are guaranteed to achieve the goal despite the non-determinism and the partial observability of the controlled system. Intuitively, a plan $\mathcal{P}$ is a weak solution to the planning problem for goal $\mathcal{G}$ from a set of states S, iff the plan is such that all the actions in it are applicable in the set of states that can be achieved by progressing the set of initial states till the current point of execution; and the set of states that can be reached by progressing the set of initial states following all the possible branches of the plan has a non-empty intersection with the set of goal states. This means that there exists an execution of the plan that can (but is not guaranteed to) reach the goal. On the other hand, strong plans are such that the execution is guaranteed to achieve the goal, despite the non-determinism of the controlled system and the incomplete run-time information.

Planning under partial observability requires being able to reason under uncertainty; *belief states* [Bonet and Geffner, 2000; Bertoli *et al.*, 2006] (i.e., non empty set of states in $\mathcal{Q}$) have been introduced to account for this. In this context, planning consists in finding a *contingent plan* that, at execution time, senses the world via observations and, depending on the its state, can conditionally execute different actions. Planning under partial observability in non-deterministic domain is an extremely hard task, and it is often the case that strong plans do not exist. However, in many cases it is possible to express reasonable assumptions over the expected dynamics of the controlled system, e.g., by identifying "nominal" behaviors. Using assumptions to constrain the search may greatly ease the planning task, allowing for an efficient construction of assumption-based solutions. Thus, assumption-based plans must be executed within reactive architectures (e.g., the one of Fig. 1) where a monitoring component traces the status of the domain,

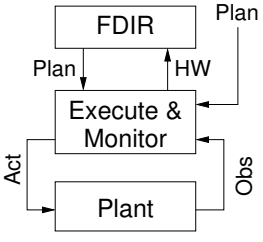

Figure 1: The approach.

in order to abort plan execution and take corrective actions activating FDIR whenever an unexpected behavior (e.g., a violation of the assumptions) has compromised the success of the plan. A safe plan also guarantees that the monitor will not trigger any plan abortion unless really needed. An example of assumption-based planning for non-deterministic, partially observable domains is given in [Albore and Bertoli, 2004; 2006]. In this work we restrict assumptions to be invariants, i.e., conditions the systems is suppose to obey at each point during the execution of the plan.

Definition 2 (Plan with Assumptions) *A plan with assumptions As is a tuple $\mathcal{P}^{As} = \langle S_g, S_{pb}, P\rangle$ where $S_g \subseteq \mathcal{Q}$ is a set of "good" states, $S_{pb} \subseteq \mathcal{Q}$ is a set of "possibly bad" states and sub-plan P is either:*

- *an empty plan ϵ;*
- *a sequence $a :: \mathcal{P}^{As}$, where $a \in \mathcal{A}$;*
- *a conditional plan $ite(o, \mathcal{P}^{As}{}_1, \mathcal{P}^{As}{}_2)$, where $o \in \mathcal{O}$.*

Sets S_g and S_{pb} are introduced to allow monitoring of the plan execution and checking if assumptions hold. The intuition is the following. Set S_g consists of those states such that the assumptions hold in them and their predecessors. Set S_{pb} includes S_g and may additionally have states indistinguishable from S_g, such that the assumptions are violated in these states or their predecessors. I.e., if during execution the assumptions always holds, then at every step the belief state has to be a subset of S_g. However, if the assumptions have been violated and this has not been detected, then the belief state may only partly intersect S_g but still has to be a subset of S_{pb}.

In the following we assume that contingent plans with assumptions are constructed by the algorithm presented in [Albore and Bertoli, 2006], simplified to deal with assumptions of type invariant. We assume that a resource assignment $\mathcal{R}_{\mathcal{MIN}}$ is given, specifying the minimal amount of resources needed during plan execution; if this limit is violated, plan execution fails. We give the following definition. Let $\mathcal{P}^{As} = \langle S_g, S_{pb}, P\rangle$ be a plan with assumptions, and let $S \subseteq \mathcal{Q}$ be a set of states. The sets of states resulting from the execution of $\mathcal{P}^{As}$ from $\langle S, \textsc{Ind}(S)\rangle$, written $\textsc{Exec}[\mathcal{P}^{As}](S, \textsc{Ind}(S))$ can be computed as follows (below we allow EXEC to be applied to plan $\mathcal{P}^{As}$ as well as to sub-plan P):

- $\textsc{Exec}[\langle \mathcal{G}_\mathcal{S}, \mathcal{B}_\mathcal{S}, P\rangle](S_G, S_{PB}) = \langle \emptyset, \emptyset\rangle$ if $S_G \not\subseteq \mathcal{G}_\mathcal{S} \vee S_{PB} \not\subseteq \mathcal{B}_\mathcal{S}$;
- $\textsc{Exec}[\langle \mathcal{G}_\mathcal{S}, \mathcal{B}_\mathcal{S}, P\rangle](S_G, S_{PB}) = \textsc{Exec}[P](S_G, S_{PB})$ if $S_G \subseteq \mathcal{G}_\mathcal{S} \wedge S_{PB} \subseteq \mathcal{B}_\mathcal{S}$;
- $\textsc{Exec}[\epsilon](S_G, S_{PB}) = \langle \emptyset, \emptyset\rangle$ if $\mathcal{R}_{\mathcal{MIN}} \not\leq_\mathcal{R} \mathcal{R}_\mathcal{E}(S_{PB})$;
- $\textsc{Exec}[\epsilon](S_G, S_{PB}) = \langle S_G, S_{PB}\rangle$ if $\mathcal{R}_{\mathcal{MIN}} \leq_\mathcal{R} \mathcal{R}_\mathcal{E}(S_{PB})$;
- $\textsc{Exec}[a :: \mathcal{P}^{As}](S_G, S_{PB}) = \langle \emptyset, \emptyset\rangle$ if $S_{PB} \not\subseteq pre(a) \vee \mathcal{R}_{\mathcal{MIN}} \not\leq_\mathcal{R} \mathcal{R}_\mathcal{E}(S_{PB})$;
- $\textsc{Exec}[a :: \mathcal{P}^{As}](S_G, S_{PB}) = \textsc{Exec}[\mathcal{P}^{As}](S'_G, S'_{PB} \cap \textsc{Ind}(S'_G))$ if $S_{PB} \subseteq pre(a) \wedge \mathcal{R}_{\mathcal{MIN}} \leq_\mathcal{R} \mathcal{R}_\mathcal{E}(S_{PB})$ where $S'_G = \{s' : s \in S_G \wedge s' \in As \wedge \langle s, a, s'\rangle \in \mathcal{T}\}$ and $S'_{PB} = \{s' : s \in S_{PB} \wedge \langle s, a, s'\rangle \in \mathcal{T}\}$;
- $\textsc{Exec}[ite(o, \mathcal{P}^{As}{}_1, \mathcal{P}^{As}{}_2)](S_G, S_{PB}) = \langle \emptyset, \emptyset\rangle$ if $\textsc{Exec}[\mathcal{P}^{As}{}_1](S_G[o, \top], S_{PB}[o, \top]) = \langle \emptyset, \emptyset\rangle \vee \textsc{Exec}[\mathcal{P}^{As}{}_2](S_G[o, \bot], S_{PB}[o, \bot]) = \langle \emptyset, \emptyset\rangle$
- $\textsc{Exec}[ite(o, \mathcal{P}^{As}{}_1, \mathcal{P}^{As}{}_2)](S_G, S_{PB}) = \langle S_G^\top \cup S_G^\bot, {S_{PB}}^\top \cup {S_{PB}}^\bot\rangle$ if $\langle S_G^\top, {S_{PB}}^\top\rangle \neq \langle \emptyset, \emptyset\rangle \wedge \langle S_G^\bot, {S_{PB}}^\bot\rangle \neq \langle \emptyset, \emptyset\rangle$ where $\langle S_G^\top, {S_{PB}}^\top\rangle = \textsc{Exec}[\mathcal{P}^{As}{}_1](S_G[o, \top], S_{PB}[o, \top])$ and $\langle S_G^\bot, {S_{PB}}^\bot\rangle = \textsc{Exec}[\mathcal{P}^{As}{}_2](S_G[o, \bot], S_{PB}[o, \bot])$

This definition is such that after plan execution, if $\langle S'_G, S'_{PB}\rangle = \textsc{Exec}[\mathcal{P}^{As}](S_G, S_{PB})$, then S'_G is still a subset of S'_{PB}. Intuitively, S'_G consists of those states whose predecessors satisfy the assumption, whereas S'_{PB} consists of those states whose predecessors may violate the assumptions but are indistinguishable from corresponding predecessors in S'_G. Thus after applying an action, the set S'_G is constrained to be a subset of the assumptions As, whereas states of S'_{PB} may violate the assumptions but have to be indistinguishable from S'_G. In the fifth and sixth items above, we check whether S_{PB} is contained in the precondition of the action a to execute, since at execution time, we cannot distinguish the states in S_G from the states in S_{PB} because of partial observability. Moreover, if the action is applicable in S_{PB} it is also applicable in S_G (since $S_G \subseteq S_{PB}$). The distinguishable states removed from S'_{PB} after applying an action (i.e., $S'_{PB} \setminus \textsc{Ind}(S'_G)$) are those which may be reached only by violating the assumptions. If such a state is indeed reached at execution time, then the plan must be terminated. Nevertheless, such plan can be considered valid since the cause of the problem is in the incorrect assumptions, not in the plan. Note that in the execution of a plan $\mathcal{P}^{As}$, the resource estimation function is computed w.r.t. the possibly bad states S_{PB}, since this is the set of states that can be observed at run-time. This choice results in considering a more pessimistic approach to the resource consumption, however more relaxed or stronger notions of validity may be considered.

The following definitions provide success criteria for plan execution. Given a resource assignment $\mathcal{R}_{\mathcal{MIN}}$ and a goal $\mathcal{G} \subseteq \mathcal{Q}$, we say that a plan $\mathcal{P}^{As}$ is *applicable* in $S \subseteq \mathcal{Q}$ iff $S_G = S \cap As$ and $S_{PB} = S \cap \textsc{Ind}(S_G)$ are non empty and the plan does not fail during execution (i.e., it does not reach empty sets): $\textsc{Exec}[\mathcal{P}^{As}](S_G, S_{PB}) \neq \langle \emptyset, \emptyset\rangle$. Moreover, $\mathcal{P}^{As}$ is *valid* in S for $\mathcal{G}$ iff it is applicable in S and for $\langle S'_G, S'_{PB}\rangle = \textsc{Exec}[\mathcal{P}](S \cap As, S \cap \textsc{Ind}(S \cap As))$:

- $S'_{PB} \subseteq \mathcal{G}$ if we want strong plan solutions;
- $S'_G \cap \mathcal{G} \neq \emptyset$ if we want weak plan solutions.

That is, for strong plans we check that the progressed set of possibly bad states S'_{PB} resulting from the execution is included in the set of goal states $\mathcal{G}$. This is because at run-time we cannot distinguish S'_G to S'_{PB} because of partial observability. Thus if S'_{PB} is included in the goal then we are guaranteed that we indeed reached the goal. On the other hand, for weak solutions we must check for non-empty intersection with S'_G. Indeed, if this is not the case the assumption was violated and the goal $\mathcal{G}$ has not been reached.

More details about plan generation, execution and monitoring can be found in [Bozzano *et al.*, 2009].

2.4 Diagnosis, Diagnosability and FDIR

The formal framework here described allows for the applicability of the techniques for tackling diagnosability, fault detection, isolation and recovery (FDIR). Diagnosis is the process of inferring the set of (most plausible) causes for an unexpected behavior of a system, given a set of observations, whereas diagnosability is the possibility for an ideal diagnoser to infer accurate and sufficient run-time information on the behavior of the observed system. The problem can be

reduced to checking whether a diagnosability condition is violated, that in turns amounts to looking for critical pairs (i.e., pairs of executions that are observationally indistinguishable, but hide conditions that should be distinguished). The problem can be solved by model checking a temporal formula representing the diagnosability condition over the coupled twin model [Cimatti *et al.*, 2003a].

Algorithm 1 Fault isolation.

```
 1: function IsolateFaults(Ass, HW, N)
 2:     Monitor := BuildFaultMonitor( );
 3:     EM := BuildProduct(M, Monitor);
 4:     R := BuildHvEqFv(M, Monitor);
 5:     R := R ∩ Ass; i := N - 1;
 6:     if 0 ≤ i then
 7:         R := R ∩ GetObs(HW[i]); i := i - 1;
 8:         while i ≥ 0 do
 9:             R := BwdImage(M, R, GetAction(HW[i]));
10:             R := R ∩ GetObs(HW[i]);
11:             R := R ∩ Ass; i := i - 1;
12:         end while
13:     FS := Project(Monitor, R);
14:     return ExtractFaults(FS);
15: end function
```

Fault detection and isolation are concerned with detecting whether a given system is malfunctioning. Fault detection analysis checks whether an observation can be considered a fault detection means for a given fault, i.e., every occurrence of the fault will eventually cause the observable to be true. Fault isolation analysis is concerned with detecting the specific cause of malfunctioning. It can be performed by generating a fault tree that contains the minimal explanations that are compatible with the observable being true. In case of perfect isolation, the fault tree contains a single cut set consisting of one fault, indicating that the fault has been identified as the cause of the malfunctioning. A fault tree with more than one cut set indicates that there may be several explanations for the malfunctioning. In this case probabilistic information can be taken into account, in order to consider the most likely explanation. For reasons of practicality, our implementation of fault isolation (see algorithm 1) only stores a limited number of the more recently performed actions and observations (the *history window*) [Williams and Nayak, 1996; Mikaelian *et al.*, 2005]. The algorithm takes the assumptions Ass under which the plan has been executed and the history window HW of size N. It starts by building a transition system that aims to monitor the value of fault variables. This monitor is then composed with the system model. Then, a bounded backward reachability of N steps from states such that the monitor variables equate the respective monitor fault variables is performed. Each step of the backward reachability is restricted to performed actions (line 9) and to the observations (line 10) stored in the history window. Finally it is restricted to the assumptions (line 11). The resulting set is projected over the monitor variables (line 13) and analyzed to extract the faults (line 14).

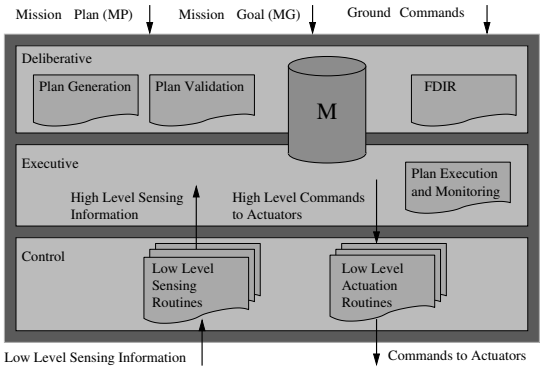

Figure 2: The ARE architecture.

3 The Autonomous Reasoning Engine

We have integrated the framework described in the previous sections within a generic autonomy architecture, called the Autonomous Reasoning Engine (ARE), developed in response to an invitation to tender of the European Space Agency aiming at developing an integrated uniform approach for model based on board autonomy [OMC-ARE, 2008]. The ARE is intended to be embedded in a spacecraft, interacting with Ground, providing autonomous reasoning capabilities, receiving information from sensors, and delivering commands to actuators.

From the logical point of view, the ARE is structured according to a generic three layers hybrid autonomy architecture (See Fig. 2). The *Deliberative Layer* is responsible for generating and validating plans, for re-planning whenever needed, and 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 the faults and to recover. Several recovery strategies can be used, depending on the degree of autonomy and fault tolerance to be achieved. If the anomaly is due to a change in the environment or in the level of resources, new assumptions can be computed, and the rest of the plan can be validated w.r.t. them. If the plan is not valid anymore, replanning can be triggered. If no recovery is possible, then a safe operational mode is entered waiting for intervention from Ground. The *Executive layer* is responsible for executing and monitoring a contingent plan coming from Ground or generated by the upper layer. The plan execution is performed accordingly to the rules described in Section 2.3. It collects performed observations and executed commands to provide them to the FDIR block for fault isolation (algorithm 1) and recovery whenever an anomaly is detected. The *Control layer* implements the conversion between the model-based level and the lower level, dealing with, e.g., sensor data acquisition and estimation of resource consumption. It is tailored to the control and monitoring of specific physical devices.

We implemented a prototype of the ARE on top of the NuSMV symbolic model checker [Cimatti *et al.*, 2000], which provides all the low level routines for the symbolic manipulation of the discrete model of the system, and the building blocks necessary for the implementation of the high

level functionalities. In addition, functionalities to upload a new model of the spacecraft, new assumptions or new mission goals are provided. We remark that 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. Currently we used a POSIX C interface, which enables easy deployment to different operating systems (e.g., RTEMS, Linux, Solaris, Windows).

4 Experimental Evaluation

We developed and validated a prototype implementation of the ARE according to ESA standards. We deployed the prototype in two industrial (a rover and a satellite) simulators (developed by Thales Alenia Space), and we run them both in SIS-ERC32 and in TSIM-LEON3, two realistic industrial hardware simulators of the ERC32 and LEON3 processors currently used in ESA.

We considered two different configurations for the planetary rover with different sub-components, each dedicated to a different space experiment. The full version consists of 20 sub-components, contains 53 state variables (encoded with 85 Boolean variables), 53 fault variables, 57 observations, and 92 commands. The small version consists 3 sub-components, contains 17 state variables (encoded with 26 Boolean variables), 15 fault variables, 19 observations, and 19 commands. The orbiter case study is simpler: is composed of 5 sub-components, contains 10 state variables (encoded with 23 Boolean variables), 5 faults, 5 observations, and 9 commands. The state space is in the range 2^{30} to 2^{200}, while the set of reachable states is in the range 2^{20} to 2^{190}.

The formal model for each case study have been specified in NuSMV starting from descriptions specified in Matlab/Stateflow/Simulink (MSS). The NuSMV models have been thoroughly validated against simulations generated from the MSS models, and against several temporal properties.

In Fig. 3 we report the time to perform the initialization of the reasoning engine w.r.t. the considered case study (i.e. the time to build the internal representation of the planning domain used by all the reasoning algorithms), the time required to build the internal representation of a mission plan generated on ground and sent on-board for being executed, the time to validate the loaded plan, the time to execute the validated plan, and finally the time required to build a new plan to achieve the goal in response to an injected fault. These times have been obtained running the ARE software on the two standard target hardware platforms used by ESA. For each of the case studies, we have identified realistic objectives, and instructed the simulator to present both nominal and anomalous conditions, in particular re-planning after a fault or an unexpected change in the environment. We have run several simulations in order to evaluate the suitability of the approach w.r.t. different metrics (e.g., unpredictable local conditions on the planetary surface, limited bandwidth, intermittent visibility, long round trip delay, rover system operations to perform a measurement cycle that included move-

	Rover (seconds)				Orbiter (seconds)	
	Small		Full			
	ERC32	LEON3	ERC32	LEON3	ERC32	LEON3
Initialization	33	13	282	113	9	1
Plan Loading	3	1	6	2	2	0.5
Plan Validation	15	6.5	55	23	1	1
Plan Execution	116	121	125	121	16	16
Plan Generation	87	34	1349	540	6	2

Figure 3: Performance characterization.

ment, sample acquisition and sample preparation and distribution). The constructed plans are strong; they have no loops, an average length of 20 actions (that depends on the goal to achieve), and an average branching factor of 1.2.

We have characterized our approach in terms of reliability (requirements coverage, generated plan compliance with the goal); availability (reaction time); and performance (processing power and memory required). The prototype was able to execute on realistic industrial hardware simulator of the ERC32/LEON3 processors actually used in ESA for space missions. The software was able to run within realistic memory limitations (32Mb) of current space applications. The performance of the approach was judged extremely positively by the domain experts in Thales Alenia Space and in ESA, both in terms of quality of the generated solutions and in terms of performance: the time taken to generate plans is order of minutes on the considered hardware (quoting ESA, "significantly smaller than the turnaround Earth/Mars communication time").

We remark that, the deployment of new technologies in real missions must always overcome many obstacles, that are not always of technical nature. Even the validation of traditional software deployed on-board is an enormous challenge, that it may also depend on the autonomy level to be achieved. For instance, the first three autonomy levels (according to the four-levels ESA classification) do not require onboard planning capabilities, but do require on-board validation and monitoring. Overall, the study has been considered by ESA as a successful (albeit small) step towards onboard autonomy. The results of the project provided inspiration for two subsequent ESA studies, one focusing on reactive execution, and one focusing on onground planning (based on the framework described here).

Additional details about the ARE software and related material are available at [OMC-ARE, 2008].

5 Related work

The first notable approach to model-based autonomy is the Deep Space One experiment by NASA. It was equipped with a Remote Agent (RA) (`http://ic.arc.nasa.gov/projects/remote-agent/`) module providing model-based execution, goal-driven planning and scheduling, diagnosis and recovery from injected faults. The model-based executive is the Livingstone model [Williams and Nayak, 1996]. Titan [Fesq *et al.*, 2002], the descendant of Livingstone, is composed of a mode estimation and reconfiguration, that is responsible for updating the current state by taking into account the commands that have been issued, and the observations perceived from the controlled system. This is performed

by considering the most likely possible state that is compatible with the history of executed actions, gathered observations, and with the system model. This approach is similar in spirit to ours, but in our case the same formal model is used in all the phases from the deliberative to the executive levels.

MUROCO-II [Kapellos, 2005] is a support tool for a generic formal framework for the specification and verification of space missions, where actions and tasks of a mission can be specified and validated. It 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-board autonomy. Second, the system developed in MUROCO-II is unable to deal with diagnosis; planning is also out of reach for all those cases where non-determinism has to be taken into account.

The MMOPS [Woods *et al.*, 2006] approach develops an on-board system, 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 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. 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

We have presented a unified model-based approach to on-board autonomy, which relies on a symbolic representation of the system under control, and supports plan generation, validation, execution, monitoring and FDIR. This approach enables the use of model checking techniques to validate the symbolic representation of the system and check for diagnosability. We have implemented our approach in a prototype of an Autonomous Reasoning Engine based on NuSMV. The experimental evaluation shows very promising results.

As future work, we plan to experiment with SAT as well as SMT techniques, to encompass both discrete and continuous components. Finally, the approach could be extended to deal with sequential goals and more complex assumptions.

References

[Albore and Bertoli, 2004] A. Albore and P. Bertoli. Generating Safe Assumption-Based Plans for Partially Observable Nondeterministic Domains. In *AAAI*, pages 495–500. AAAI, 2004.

[Albore and Bertoli, 2006] A. Albore and P. Bertoli. Safe LTL Assumption-Based Planning. In *Proc. International Conference on Planning and Scheduling*, pages 193–202. AAAI, 2006.

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

[Biere *et al.*, 2003] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[Bonet and Geffner, 2000] B. Bonet and H. Geffner. Planning with Incomplete Information as Heuristic Search in Belief Space. In *Proc. International Conference on Planning and Scheduling*, pages 52–61. AAAI, April 2000.

[Bozzano *et al.*, 2009] M. Bozzano, A. Cimatti, M. Roveri, and A. Tchaltsev. A Comprehensive Approach to On-Board Autonomy Verification and Validation. In *Verification & Validation of Planning and Scheduling Systems*, Thessaloniki, Grece, 2009.

[Bryant, 1992] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Comp. Surveys*, 24(3):293–318, September 1992.

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

[Cimatti *et al.*, 2003a] A. Cimatti, C. Pecheur, and R. Cavada. Formal verification of diagnosability via symbolic model checking. In G. Gottlob and T. Walsh, editors, *IJCAI*, pages 363–369, 2003.

[Cimatti *et al.*, 2003b] 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.

[Clarke *et al.*, 1999] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999. ISBN 0-262-03270-7.

[Clarke *et al.*, 2003] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[Eisner and Fisman, 2006] C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Springer-Verlag, 2006.

[Fesq *et al.*, 2002] 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, 2002.

[Ghallab *et al.*, 2005] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufman, 2005.

[Kapellos, 2005] K. Kapellos. Formal Robotic Mission Inspection and Debugging (MUROCO II) Executive Summary, Issue 1, ESA Contract 17987/03/NL/SFe, 2005.

[Lahiri *et al.*, 2006] S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras. SMT techniques for fast predicate abstraction. In *CAV 2006*, volume 4144 of *LNCS*, pages 424–437. Springer, 2006.

[Mikaelian *et al.*, 2005] T. Mikaelian, B. C. Williams, and M. Sachenbacher. Model-based monitoring and diagnosis of systems with software-extended behavior. In *AAAI*, pages 327–333. AAAI Press, 2005.

[OMC-ARE, 2008] OMC-ARE: On Board Model Checking - Autonomous Reasoning Engine, 2008. `http://es.fbk.eu/projects/esa_omc-are/`.

[S. Graf and H. Saidi, 1997] S. Graf and H. Saidi. Construction of Abstract State Graphs with PVS. In *CAV'97*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

[Williams and Nayak, 1996] 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.

[Woods *et al.*, 2006] M. Woods, D. Long, R. Aylett, L. Baldwin, and G. Wilson. Mars Mission On-Board Planner and Scheduler (MMOPS) Summary Report, Issue 1, ESA Contract 17987/03/NL/SFe CCN1, 2006.