Probabilistic Model Checking applied to Autonomous Spacecraft Reconfiguration

Vittoria Nardone*, Antonella Santone*, Massimo Tipaldi†, Luigi Glielmo*
* Department of Engineering, University of Sannio, Benevento, Italy
e-mail: {nardone, santone, mtipaldi, glielmo}@unisannio.it
† Compagnia Generale per lo Spazio, Via Gallarate 150, 20151 Milano, Italy
e-mail: mtipaldi@cgspace.it

Abstract—Formal verification techniques are necessary to demonstrate the completeness, the correctness, and the consistency in implementing spacecraft model-based autonomy requirements, where we have to explore the behaviour of the system over a large input range. This paper introduces a methodology based on probabilistic model checking to verify the specification of spacecraft autonomous reconfiguration functionality, modeled via Markov Decision Process. Some preliminary results show how this approach can be used to check whether the system fulfills suitably-defined properties under a given probability.

Keywords—Model Checking, Probabilistic Computation Tree Logic, Spacecraft Autonomy, Markov Decision Process.

I. INTRODUCTION

A large number of surveys have been conducted on the subject of software (SW) testing practices in different countries and scales [1]. This denotes an increasing interest towards SW testing activities in the industry, including their connection to SW quality requirements. Many SW practitioners show their awareness in test process definition and improvement, testing automation, testing tools, and standardization. These aspects are particularly true in case of critical space SW applications (such as the ones running on-board of spacecrafts) which require high quality development standards. The total effort in SW Verification and Validation (V&V) activities is considerably higher than the one spent in SW detailed design and implementation.

The ECSS-E-ST-40C [2] offers an exhaustive description of the SW V&V activities currently adopted in space projects. They include SW requirement consistency & completeness analysis, SW unit tests, code inspection, static code analysis, SW metrics collection and analysis, SW/SW integration tests, SW functional and performance tests, and spacecraft-level tests. SW V&V activities are usually scaled according to the associated SW criticality levels, ranging from Level A (software whose anomalous behaviour would cause or contribute to a failure resulting in a catastrophic event) up to Level E (software whose anomalous behaviour would cause or contribute to a failure resulting in a negligible event).

In case of model-based autonomy requirements in space missions, traditional V&V approaches are not sufficient to demonstrate the completeness, the correctness, and the adequacy of related SW artifacts. Since autonomous systems are sensitive to the environment and the actual mission environment cannot be predicted with sufficient accuracy, one should explore the behaviour of the system over a range of plausible inputs in order to demonstrate its robustness [3]. This makes the traditional V&V approach unfeasible and paves the way for completely different solutions. Formal or analytical methods can provide effective solutions. In general, they refer to the use of techniques from logic and discrete mathematics and can be categorized as follows [4]: runtime monitoring, static analysis, model checking, theorem proving, and compositional verification.

This paper introduces a methodology based on probabilistic model checking to support the verification of spacecraft model-based autonomy requirements, in particular related to spacecraft autonomous reconfiguration. Model checking [5] evaluates the system behaviour and its state evolution by means of its representative model and a set of properties expressed via an executable specification language. Basically model checkers require two inputs: a description of a model, represented as a state transition system, and a specification, typically a formula in some temporal logic, and return yes or no, indicating whether or not the model satisfies the specification. Probabilistic model checking is a generalization of these techniques, aimed at systems whose behaviour is stochastic in nature. Thus, the models are probabilistic, in the sense that they encode the probability of making a transition between states instead of the simple existence of such a transition. Moreover, the analysis normally entails calculation of the actual likelihoods through appropriate numerical or analytical methods.

We show how to use the probabilistic model checking to spacecraft autonomous reconfiguration, by modeling its requirements into PRISM [6] modules, and then checking whether they satisfy suitably-defined properties with a given probability. PRISM is an open-source probabilistic model checker that provides support for verifying and analyzing several types of probabilistic models: discrete- and continuous-time Markov chains, Markov decision processes, and extensions of these models with rewards. Properties are expressed using the Probabilistic Computation Tree Logic (PCTL*) [7].

According to our current knowledge, we have exploited the capabilities of probabilistic model checking in the aerospace context in a rather unexplored area with encouraging results. In any case, this paper presents some preliminary results that have to be consolidated on the methodology side in order to make it applicable to more realistic scenarios.
The remainder of this paper is organized as follows. First, in Section II we provide an overview on spacecraft on-board autonomy, probabilistic model checking, and PRISM. In Section III, we show how formal probabilistic model checking can be profitably applied to verify autonomous spacecraft reconfiguration properties. Finally, Section IV concludes the paper and provides some pointers to future research activities.

II. PRELIMINARIES

In this section, we introduce some concepts on autonomy in space missions. Then, we quickly recall some main aspects concerning the probabilistic model checking and PRISM [6].

A. Autonomy in Space Missions

Autonomy is an important feature of currently developed and future missions [8]. The traditional notion of autonomy as predefined explicit behaviours can be adequate for satellites operating in predictable environments where activities can be determined in advance. However, this approach breaks down under increasing context uncertainty, which features deep space exploration systems or critical operational phases such as automated spacecraft docking. In the near future, these systems will be able to receive, process and achieve high-level goals even in an uncertain or dynamically varying context. They will be endowed with an ergonomic interface so that ground operators can easily define and transmit such high level requests. Ground high-level objectives can be further on-board detailed into a sequence of commands for the subsystems and can be autonomously adapted during their execution according to context changes, such as altered on-board resource profile.

From an operational point of view, on-board autonomy can be regarded as migration of functionality from the ground segment to the flight segment. In [8], four broad application fields have been identified: intelligent sensing, model-based fault protection, distributed decision-making, and planning and execution. The latter, addressed in this paper, concerns the process of decomposing, and then executing, high-level goals into a sequence of activities that satisfy temporal and resource constraints. The On-Board Software (OBSW) plays a relevant role in implementing on-board autonomous capabilities. The integration of planning systems and dynamic reprogramming capabilities into the flight software has been recalled in [3] where the OBSW architecture has been organized along three hierarchical levels:

- Decisional level: it is in charge of programming the activities of the platform and/or payload, and monitoring the execution of the activity plan.
- Operational level: it is in charge of the execution of the operations (decomposition and routing of commands), based either on the activities decided in the upper level or on commands sent by ground via the ground/space interface.
- Functional level: it controls and supervises the various spacecraft subsystems (e.g. Power Electrical Subsystem) by executing commands coming from the operational level and performing subsystem level monitoring.

Spacecraft autonomous reconfigurability requirements can be implemented at decisional level. An interesting solution via an MDP based framework is shown in [9], where the optimal policy is calculated by taking into account not only the uncertainty in the detection of faults, but also the currently active mission related actions and remaining objectives of the mission. The model implemented in this paper for model checking verification is inspired by the approach described in [9], with some customization needed to implement it at decisional level.

B. Probabilistic Model Checking and PRISM

Model checking is a technique to establish the correctness of complex systems in an automated fashion. Model checkers require two inputs: a description of a model, typically a finite state transition system, and a specification, typically in a temporal logic form, and return yes or no, indicating whether or not the model satisfies the specification. In the case of probabilistic model checking, the models are probabilistic, in the sense that they encode the probability of making a transition between states instead of the simple existence of such a transition, and the analysis normally entails calculation of the actual likelihoods through appropriate numerical or analytical methods [10]. In this paper we use the Markov decision process (MDP), which is one of the most common formalisms for modeling systems with both probabilistic and non-deterministic behaviour.

A State Labeled Markov Decision Process (MDP) is a tuple

\[
\mathcal{M} = (S, \pi, A, \tau, \mathcal{L})
\]

where

- \( S \) is a \((finite)\) set of states;
- \( \pi \in S \) is an initial state;
- \( A \) is a \((finite)\) set of actions;
- \( \tau : S \times A \times S \to [0, 1] \) is a likelihood transition function such that, for each state \( s \) and enabled action \( a \), \( \tau(s, a, s') \) gives the probability of taking action \( a \) in state \( s \) and moving to state \( s' \); and
- \( \mathcal{L} : S \to 2^{AP} \) is a labelling function mapping each state to the set of atomic propositions \((AP)\) true in that state.

More precisely, for \( s \in S \) , \( a \in A \), either \( \sum_{s' \in S} \tau(s, a, s') = 1 \) (\( a \) is enabled) or \( \sum_{s' \in S} \tau(s, a, s') = 0 \) (\( a \) is disabled). For each \( s \in S \) there exists at least one action enabled from \( s \).

A probabilistic model checker tool automates the correctness proving process. A popular probabilistic model checker is PRISM [6], successfully adopted in many application domains such as communication and multimedia protocols, randomized distributed algorithms, security protocols, biological systems [11]. In PRISM the system model is defined with a probabilistic reactive module and the system behaviours are defined in terms of properties written in PCTL* [7], a probabilistic extension of the temporal logic CTL. More precisely, a PRISM model is composed of modules, whose state is determined by a set of variables and whose behaviour is specified by a set of guarded commands. A guarded command contains an
Fig. 1: The workflow of the approach.

(omitted action label, a guard variable, and a probabilistic
update definition for the module variables:

\[ \text{[action]} \text{ guard} \rightarrow \text{prob1 : update1 + ... +}
\]
\[ \text{probn : update;} \]

When a module has a command whose guard is satisfied
in the current state, it can update its variables probabilisti-
cally, accordingly to the update definition. For action-labeled
commands, multiple modules execute updates synchronously,
if all their guards are satisfied. Each probabilistic transition
in the model is thus associated with either an action label
or a single module. A scheduler for a MDP resolves the non-
determinism in each state \( s \) by providing a distribution over the
set of actions enabled in \( s \). System behaviours are defined in
terms of properties written in PCTL*. A fundamental feature
of PCTL\(^*\) logic is the probabilistic operator \( P \), which allows
one to reason about the probability that executions of the
system satisfy some property. For example, the formula:
\[ P_{max=?}[F \text{ lost}] : \text{returns "the maximum probability, across}
\]
\[ \text{all possible schedulers, of the protocol losing a message".} \]

III. THE METHODOLOGY

In this section we show how formal probabilistic model
checking can be profitably applied to verify autonomous
spacecraft reconfiguration properties. More precisely, from
the spacecraft autonomy requirements we can derive PRISM
modules, which are successively used to perform formal ver-
ification, as can be seen in Figure 1. Three steps are required
by our process:

- PRISM Model Definition;
- Properties Definition;
- Probabilistic Verification.

In the following subsections the three steps are discussed in
detail.

A. PRISM Model Definition

We use as internal representation the PRISM language. In
this section we present a model describing the evolution of a
simplified spacecraft through its operational modes. We define
two PRISM modules:

- **Spacecraft**: it models the spacecraft configuration
  changes for each spacecraft mode;
- **Event**: it describes the events, i.e. anomalies that can
  occur during the spacecraft mission.

Considering the unpredictable nature of a spacecraft con-
text, the evolution of its modes and configurations has been
modeled via an MDP process. We adopt the non-determinism
to select the initial configuration. The model evolves in the
following way: starting from an initial configuration, the
**Spacecraft** module remains in the same status with \( (1 - \Theta) \)
probability (where \( \Theta \) is specified in the model), whereas the
system changes its configuration with a \( \Theta \) probability. In
the **Spacecraft** module three spacecraft modes have been
considered and are represented by the variable \( a \):

- **0**: Stand-by Mode. The spacecraft acts as an in-orbit spare
  not used for any operational service but able to become
  operational within a short time frame. Full health and
  safety activities are maintained, including continuous real
time monitoring.
- **1**: Attitude Maneuver Mode. The spacecraft activates
  some control laws and mechanisms for attitude maneu-
  vering and stabilization.
- **2**: Collection Science Data Mode. The spacecraft retrieves
  and processes scientific data of relevant importance to
  fulfill the overall mission objective.

These two PRISM modules (**Spacecraft** and **Event**)
evolve concurrently. When an exogenous event occurs, in
order to solve the anomaly, the spacecraft tries to apply a
specific configuration. In the light of the above described events, the system can reach the suitable configuration in order to handle the occurred event. As preliminary concept, we assume that events of a specific degree are solved by the configuration with same degree (i.e., the event $i$ is handled by configuration $i$, for $i \in [1..3]$). Moreover, the case of unresolved anomalies has been specified in the model as well, see Figure 2 for the module Event PRISM specification.

B. Properties Definition

To verify the behaviour of our model, we have to specify some path properties. A path property is a formula that can be evaluated either true or false for a single path in a model. Since we use an MDP model, we have two possible types of properties computing either maximum or minimum probability values. As an example we mention two properties that can be verified on the model. Following we show these properties, and we retrieve two versions: the first one computes the maximum probability and the second one the minimum.

\[
\begin{align*}
\text{Pmax} = \text{Pmax=?} \quad \text{(Property 1)}
\end{align*}
\]

\[
\begin{align*}
\text{Pmin} = \text{Pmin=?} \quad \text{(Property 2)}
\end{align*}
\]

\[
\begin{align*}
\text{Pmax} = \text{Pmax=?} \quad \text{(Property 3)}
\end{align*}
\]

\[
\begin{align*}
\text{Pmin} = \text{Pmin=?} \quad \text{(Property 4)}
\end{align*}
\]

As shown, the maximum probability that a low severity event is solved (in the next state of its occurrence) is equal to 1, meaning that the resolution of this kind of event is always guaranteed in the next step. Otherwise the minimum probability that a low severity event is solved in the next step is equal to 0.25. This result is coherent with the previous one as it represents the same property, the only difference is in the computing (Pmax vs Pmin). Differently for the second global property: the maximum probability that a low severity event increases its severity degree in the next step is equal to 0.3749, instead the minimum probability that the above behaviour happens is equal to 0.

Starting from these preliminary results, we state that the spacecraft is always able to solve the low level anomalies. However, anomalies of the same or higher severity degrees can occur in the next step.

Even if the proposed novel approach for the verification of the reconfiguration capabilities in Space SW is advantageous, it is worth noting that only model-based verification cannot be sufficient, because if the model is flawed it might lead to false confidence on the system. Thus, the “traditional” testing comes as supporting (even though not exhaustive) evidence confirming the model-based checking results.

IV. Conclusion and Future Work

In this paper we have investigated a methodology based on probabilistic model checking to verify the specification of spacecraft autonomous reconfiguration functionality, modeled via Markov Decision Process. PRISM tool has been used to model its stochastic state transition dynamics, whereas its properties to be verified have been formulated by means of the Probabilistic Computation Tree Logic. Some preliminary results show how this approach can be used to check whether the system fulfills suitably-defined properties under a given probability. The applicability of this methodology has to be further investigated and consolidated in order to make it
applicable to more realistic scenarios, where the number of system variables and properties can be considerably higher. Interesting direction can be the property verification of an autonomous spacecraft driven by sub-optimal policies calculated via Approximate Dynamic Programming techniques.

It is well-known that formal methods cannot be easily scaled due to the state explosion problem, which says that the state space grows exponentially in the number of concurrent processes. In fact, the parallelism between the processes of the system leads to a number of reachable states which may become very large, in some cases on the order of millions or billions of states. When the number of states is too large to fit in a computer’s main memory, verification quickly breaks down. Several approaches have been developed to solve or reduce the state explosion problem. We are planning to apply some reduction techniques developed by one of the author and al. to combat the state explosion problem, see [12], [13], [14], [15], [16], [17]. These techniques have been proposed for process algebras, thus it will be interesting to migrate them to probabilistic model checking.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the contribution of Enrico Di Cerbo for his help in implementing the PRISM models and producing the simulation results.

REFERENCES


Fig. 3: Model simulation for a 100s time window.