Quantitative Risk Assessment in the Design of Resilient Systems

Braham Lotfi Mediouni, Iulia Dragomir, Ayoub Nouri and Saddek Bensalem

Verimag Research Report n^o TR-2018-10

December 17, 2018

Reports are downloadable at the following address
http://www-verimag.imag.fr

Unité Mixte de Recherche 5104 CNRS - Grenoble INP - UGA
Bâtiment IMAG
Université Grenoble Alpes
700, avenue centrale
38401 Saint Martin d'Hères
France
tel : +33 4 57 42 22 42
fax : +33 4 57 42 22 22
http://www-verimag.imag.fr/

Quantitative Risk Assessment in the Design of Resilient Systems*

Braham Lotfi Mediouni(✉), Iulia Dragomir(✉), Ayoub Nouri(✉), and Saddek Bensalem(✉)

Univ. Grenoble Alpes, CNRS, Grenoble INP**, VERIMAG, 38000 Grenoble, France
firstname.lastname@univ-grenoble-alpes.fr

Abstract. Deploying fault detection, isolation and recovery (FDIR) subsystems is an accepted solution to address the occurrence of faults and failures in safety-critical (real-time) systems. Yet, these FDIR subsystems should be devised only for those faults that falsify the system's requirements. As a consequence, the obtained system is minimal, although complete, and robust both with respect to safety and performance requirements. In this paper we propose a two-fold systematic and mechanized approach based on formal methods combining (1) the evaluation of faults relevance based on quantitative risk assessment, and (2) the validation of system robustness by statistical model checking. We apply this approach on an excerpt of a real-life autonomous robotics case study, and we report on the implementation and results obtained with the $\mathcal{S}$BIP framework.

Keywords: Model-based system design · FDIR · Risk assessment · Statistical model-checking · Real-time systems · $\mathcal{S}$BIP framework · Robotics case study.

1 Introduction

The correct system design is in general a hard problem that depends on many factors such as system complexity, requirements satisfaction, tool-chain constraints, etc. In the case of real-time safety-critical systems, uncertainties at runtime also need to be taken into account at design time. Indeed, these situations may have serious implications on the system's execution and mission. Corrective measures to handle them after the system deployment can prove to be very costly, and in some cases impossible to implement.

The type of uncertainties considered in this paper are faults and failures that occur at system execution. To address these cases and allow for resilience, systems are equipped with *fault detection, isolation and recovery* (*FDIR*) capabilities. FDIR is usually designed as software components that detect whether a fault has happened and apply a predefined sequence of actions that bring the system in a safe mode. The current practice for designing FDIR components follows an ad-hoc process based on the engineers expertise. This process takes into account requirements expressed in natural language and produces implementations that are integrated in the system. We identify several difficulties within this process mainly related to its completeness and automation: (i) faults are identified from informal specification and requirements, (ii) the impact of faults is evaluated manually, and (iii) FDIR implementations are validated by unitary and integration testing with no formal guarantees.

Formal methods have been recently leveraged for correct-by-construction FDIR components [40] in the frame of untimed [13] and real-time [19] systems. In this context, synthesis algorithms are used for building the two parts of the FDIR components: the diagnoser for the fault detection and the controller for the recovery. Two limitations are identified in [19] with respect to the proposed approach: (iv) a diagnoser is devised for all detectable faults, and (v) the controller is manually modeled being left for verification by model-checking techniques. Firstly, synthesizing a diagnoser for each detectable fault has inconveniences since not all faults have an impact on the requirements to satisfy, and the system analysis and performance can be greatly degraded due to the large number of unnecessary components. Therefore, it is important to synthesize diagnosers only for those faults that are relevant with respect to the system requirements and

* This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement #730080 (ESROCOS), #730086 (ERGO) and #700665 (CITADEL).

** Institute of Engineering Univ. Grenoble Alpes

objectives. This limitation is emphasized by the manual activity of evaluating faults, as described by item (ii) above. Secondly, the controller validation problem is hard and often unfeasible since model-checking techniques suffer from scalability issues.

In this paper we tackle the limitations described above by proposing a model-based development approach that relies on quantitative risk assessment and formal methods. The aim of this approach is to design systems resilient to faults in an incremental manner based on model transformation. We consider risk to be any system-related changes that may alter the system nominal behavior or its performance. In our approach, risk is introduced through model transformation, explicitly by modeling faults and implicitly by integrating new FDIR capabilities. Quantitative risk assessment is used to study the impact of such changes and to improve the FDIR design. To tackle limitations (ii) and (iv), we use probabilities to automatically measure risk and to evaluate whether it is tolerable or not. When risk is deemed unacceptable, mitigation would be either to synthesize a new FDIR component or to enhance the existing ones, e.g., with a more appropriate recovery strategy.

In this work, we automate probabilistic risk measurement by using *statistical model-checking* (*SMC*) [22,41] and we leverage its scalability to validate manually designed controller components as described by limitation (v). SMC is a formal verification method that combines simulation with statistical reasoning to provide quantitative answers on whether a stochastic system satisfies some requirements. SMC is widely used in various domains such as biology [15], communication protocols [8], multimedia [37] and avionics [9]. It has the advantage to be applicable to models and implementations provided that they meet specific assumptions, in addition to capture rare events.

More precisely, the contributions of this paper are the following:

- The definition of an iterative and incremental process for the design of resilient systems equipped with FDIR capabilities. This process is model-based and integrates quantitative risk assessment and system validation which are partially automated using SMC as described in Section 4.
- The application of this design process on a real-life robotics case study. We devise three system designs at different levels of granularity on which we perform quantitative risk assessment. For each design we propose FDIR behavior that we validate against the system's requirements. We use the $\mathcal{S}$BIP formal framework [29,33] described in Section 3 for modeling and quantitative analysis. The obtained results are presented in Section 5 and 6.
- A discussion of the advantages and limitations of this process in the design of industrial applications is given in Section 7.

2 Related Work

The risk asssessment activity defined in [23] can be applied to different application domains, including computer-based systems. A survey about the challenges and current practices for risk assessment in computer-based systems is presented in [42]. The literature tackles this activity mainly from two different aspects: *security* and *safety* risk assessment. In the context of security risk assessment approaches are proposed in [6,14,26], while [28,38,20,35,30] focus on the representation of risks in the system design and the corresponding automated analysis methods. For example, [30] considers risks as attack scenarios and statistical model-checking is used to perform risk analysis.

Safety risk assessment is studied from two points of view: qualitative or quantitative. Qualitative safety assessment determines what scenarios lead the system from a nominal mode to a degraded mode where safety requirements do not hold. The practice consists of building safety artifacts such as fault trees or timed failures propagation graphs and analyzing them in order to certify the system safety. For example, automated safety analysis for fault trees is described in [10] for the AltaRica dataflow language. In [12] the xSAP tool is presented for the analysis of fault trees and timed failure propagation graphs in the context of symbolic transition systems à la nuXmv.

Quantitative safety assessment provides probabilistic measures of the risks in the systems, such as the likelihood of failure. Probabilistic computations are usually done manually on the safety artifacts build a

priori, on which the probability distributions for faults are added. Some safety assessment tools automate this analysis, such as xSAP for probabilistic fault trees. The work presented in this paper contributes to this class of risk assessment methods. In our case we use SMC in order to compute the probability for the system to fail as described by its requirements.

The rationale to use SMC is feature-based, as briefly presented in Section 1. Firstly, compared to other formal verification techniques such as (probabilistic) model checking [7], SMC is *scalable*. This feature is inherent to the method: only a subset of the system's executions are explored, while the underlying statistical algorithms can be easily parallelized. Even though the obtained results are only estimations, their accuracy is controlled with confidence parameters that bound the estimation error, and which distinguish it from pure simulation techniques. Also, corner cases and rare events which might be missed by simulations can be caught with SMC. Specific techniques such as *importance sampling* and *importance splitting* [25,24] have been recently adapted to SMC in order to efficiently deal with this class of events. Another important feature of SMC is its usability on both models and implementations, provided that implementations are obtained from formally defined models with a purely stochastic semantics and code generation preserves the semantics.

Safety risk assessment can be seen as an optimization in the design of FDIR components in general and diagnosers in particular, as explained in Section 1. The correct design of FDIR components from complete system specifications has been studied from methodological point of view in [40,13,19]. Implementations are provided in [19] for timed systems with partial observability and in [13] for untimed systems à la nuXmv. While [13] includes the safety assessment mechanism implemented in [12] for user-modeled timed failure propagation graphs, this question is left open in [19]. Our contribution completes the work from [19] by defining and automating a quantitative safety assessment method for (stochastic) timed systems allowing the efficient design of FDIR components.

Finally, our case study tackles the rigorous design of a robotics control system. In [11,3], (RT)BIP and (RT)DFinder tool are used to model and verify safety and performance requirements such as causality in service executions, mutual exclusion and data freshness. The TINA model checker is used in [21] to verify schedulability properties of a robotics application tasks on a given hardware platform. In [39] safety properties for modular robots such as conflicting commands, self-collision, etc., are checked and simulated for different configurations. Diagnosers are implemented using formal models for robotics control software in [17] with respect to safety properties and using the P language. In [31], differential dynamic logic is used to model and verify the behavior of ground robots, while a diagnoser to ensure the nominal behavior is synthesized with ModelPlex and added to the system. The contribution presented in this paper has been used for the development of the FDIR components in the robotics systems scenarios presented in [32,34].

To the best of our knowledge, this work is the first to use statistical model-checking for quantitative risk assessment in the design of resilient systems in general, and for FDIR behavior in particular.

3 A Rigorous Framework for Modeling and Analyzing Stochastic Timed Systems

Stochastic models are of paramount importance in system design as they allow to capture uncertainties, a key concept for reasoning about risk. Besides, models of real-time behavior are mandatory when designing critical applications. To take full advantage of these models, formal and quantitative analysis techniques allowing to handle real-life system models are primordial.

In this work, we consider SMC which takes as input an executable model of the system of interest and a formal specification of the requirement to verify, usually given in some logic. Using SMC, it is possible to (1) estimate the probability that the system satisfies the requirement, or (2) position the probability of satisfying the requirement with respect to a given threshold. Answering the first type of query relies on well known probability estimation techniques [22], while the second is handled using a hypothesis testing approach [41]. Concretely, SMC explores a sample of finite execution traces which are iteratively generated and monitored against the desired requirement. Monitored traces produce local verdicts $\{true, false\}$ which are consumed sequentially by the statistical algorithms to compute a final probability estimation or a global verdict.

In this section, we briefly present the $\mathcal{S}$BIP framework [29,33] that represents the foundation on which relies our approach. It includes a stochastic real-time modeling formalism and a statistical model checking engine.

3.1 Stochastic Real-time BIP

The stochastic real-time BIP [33] is a component-based modeling formalism that allows to construct complex system models compositionally. This formalism is sufficiently expressive to model systems from various application domains, including different behaviors such as real-time, uncertainties and faults. It further allows to integrate external code in order to model complex computations.

In the stochastic real-time BIP formalism, components are designed as extended timed automata [5] and are composed through multi-party interactions, i.e., n-ary synchronization among components actions. Before formally defining components, we introduce some notation.

Let X be a finite set of variables called clocks. We denote by $\Phi(X)$ the set of convex constraints on X given by the grammar: $\varphi ::= true | x \sim c | \varphi \wedge \varphi$, with $c \in \mathbb{Q}$ and $\sim \in \{<, \leq, =, >, \geq\}$. Similarly, we define by V a finite set of discrete typed variables, and with $\Phi(V)$ first-order expressions (without quantifiers) on variables from V.

Definition 1 (Stochastic Real-time BIP component). *A component is an extended timed automaton* $\langle L, \ell_0, X, V, I, P, U, F, T \rangle$*, where:*

- L *is a finite set of locations,*
- $\ell_0 \in L$ *is the initial location,*
- X *is a finite set of clocks,*
- V *is a finite set of discrete variables,*
- $I : L \to \Phi(X)$ *is a function associating to each location some clock constraint,*
- P *is a finite set of ports, and*
- U *is the set of urgencies* $\{\epsilon, d, \lambda\}$*,*
- F *is the set of update functions on* X *and* V*,*
- T *is a finite set of transitions. Transitions are of the form* $(\ell, p, g, u, f, \ell')$*, where* $\ell, \ell' \in L$ *are the source and target locations,* $p \in P$ *is the triggering event,* $g \in \Phi(X) \cup \Phi(V)$ *is the guard,* $f \in F$ *is the update function, and* $u \in U$ *is the urgency.*

A component is a finite automaton enriched with data and real-valued clocks that allow to measure time delays. Time elapse is restricted in each location with a clock constraint, as modeled with I in the definition above. A transition can be fired when its guard is enabled, that is the valuation of clocks and data satisfy the constraint. Additionally, time elapse is controlled on transitions with urgencies, namely *eager* (ϵ), *delayable* (d) or *lazy* (λ). Eager specifies that the transition must be fired as soon as it is enabled. Delayable states that the transition can be delayed at most to the upper bound of the time interval. Finally, lazy specifies that the event can be fired at any moment while enabled or never.

Uncertainty in the stochastic real-time BIP formalism concerns mainly events scheduling. It is expressed by associating events guards with probability density functions. Hence, the precise moment of executing an event is scheduled according to that density. We consider two types of events, namely *timed* and *stochastic*. The former are associated to timing constraints on transitions. These events are implicitly scheduled according to a uniform or an exponential probability distribution as it is the case in several existing modeling formalisms e.g., Uppaal [16]. *Stochastic events* can be associated to arbitrary density functions, e.g., Normal or Poisson, and scheduled accordingly. The underlying semantics of a stochastic real-time BIP model is a Generalized Semi-Markov Process (GSMP) [27] where the interpretation of time is dense[1].

[1] We refer the readers to [33] for the formal definition of the stochastic real-time BIP.

3.2 The SMC-BIP Engine

SMC-BIP [29] considers as input stochastic real-time BIP models and requirements expressed in Linear-Time Temporal Logic (LTL) [36] and Metric Temporal Logic (MTL) [4]. It implements both types of SMC queries in addition to advanced features such as automatic parameters exploration, useful for system dimensioning, and rare events analysis, important for risk assessment. The tool offers an integrated development environment including a graphical user-interface permitting to edit, compile and simulate models, and automates the different statistical analyses. In the context of this work, we mainly use the probability estimation and parameter exploration capabilities offered by the tool.

4 A Model-based Approach Integrating Quantitative Risk Assessment

The proposed approach, illustrated in Figure 1, is based on idea of iterative and incremental transformation of models Γ. Each model transformation can introduce new risks, for example due to relaxing environment assumptions. The idea depicted in this approach is to perform at each step of the development two assessments. First, *quantiative risk assessment* allows to measure the impact different risks have with regard to the system requirements, and perform a model upgrade if deemed necessary. Secondly, *validation* ensures that the upgrade is consistent with respect to the system requirements. Please note that the proposed approach is general enough to be applied to different types of systems, e.g., untimed, real-time. Moreover, the notion of *risk* can have different interpretations, e.g., safety, security. Our setting consists of stochastic real-time systems designed and analyzed with the $\mathcal{S}$BIP framework, where risks are understood and modeled as faults.

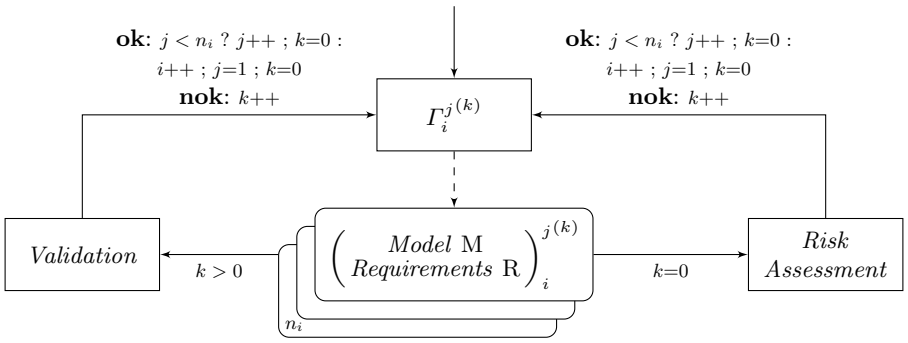


Fig. 1: Design approach based on formal methods integrating quantitative risk assessment where: Γ denotes model transformation, i is the index of the number of performed steps, j is the index for the number of explored models within a step bounded by n_i, and k is the number of iterative transformations performed on a model. Initially i is set to 0, and j and k to 1.

Initially, system specifications and informal general requirements are analyzed ($\Gamma_0^{1\,1}$) to build a nominal application model ($M_0^{1\,(1)}$) and a set of formal requirements ($R_0^{1\,(1)}$). The only assessment performed at this step is the validation (i.e., $k = 1$): the model should satisfy the formal requirements. While this condition is not satisfied, the model is iteratively transformed, as denoted by the **nok** label and index k in Figure 1. When the model is judged valid, one can proceed with the next model transformation step.

The model is incrementally transformed towards the concrete implementation as represented by the i index. Transformation concerns different aspects of the system and may introduce new risks. Transformation examples include integrating new behavior, correction of bugs in the model or legacy code, instantiation of the model's parameters. For the latter, one obtains a family of models indexed with $j \in \{1, \ldots, n_i\}$ in Figure 1. Similarly, the system requirements are modified based on the purpose of the performed model transformations. By system requirements we mean those expected to be fulfilled by the system model in the current stage of the design, that is, during the step i the exploration j and the iteration k.

The first analysis to perform on the system model $M_i^{j(0)}$ is the *risk assessment*. It implies computing the probability for requirements $R_i^{j(0)}$ to hold on the model. Based on this measurement, the risk can be appreciated. If they are acceptable, represented with the **ok** label, one can continue with inspecting a new model either from the same family if $j < n_i$ or by moving to the next step $i+1$. If the risks are high, represented with the **nok** label, a *decision* on how to mitigate is taken, which usually involves the transformation of the model architecture and/or behavior. Once all the risks have been dealt with, the obtained model $M_i^{j(1)}$ and its requirements are subject to the iterative validation described above.

In the following sections we show how to apply this approach on a robotics control system case study. We distinguish four levels of granularity for this case study. At level 0, we design a nominal application model that we validate with respect to initial requirements. At level 1, we introduce risks in the form of faults and perform risk assessment. The decision consists of introducing FDIR functionality in one of the components of the model, which we then validate. At level 2, a performance-related model transformation is applied that impacts only the set of requirements. We show that the FDIR behavior introduced is necessary but not sufficient with respect to performance and we propose an improvement that is again validated. Finally, at level 3, we introduce in the design the deployment model. We instantiate the deployed model that considers 3 aspects and we explore several deployments using risk assessment with respect to performance-specific requirements. All models are described in the stochastic real-time BIP formalism where time evolves probabilistically following uniform density functions. The risk assessment and validation activities are automated using the SMC-BIP tool.

5 Planetary Robotics Case Study

The case study considered in this paper is an excerpt of a Bridget Rover demonstrator control system developed for the validation of the ESROCOS environment [2]. The Bridget Rover (see Figure 2) is a representative of Martian rovers aiming for planetary exploration while providing a modular and configurable payload bay. The payload consists at least of a panoramic camera for image acquisition aiming for biological signatures detection and autonomous driving via map construction. The rover uses 6 wheels to drive and steer, where the motors communicate with the locomotion system via a CAN-bus.

5.1 System and Requirements Overview

The control system developed with ESROCOS [1] aims to remotely drive the rover using a joystick and acquire images. In consequence, the developed system interfaces with the low level control of the locomotion system and the CAN-bus node. We consider in the following the drive with a joystick functionality of the developed control system as case study [18].

More precisely, the case study consists of the software chain between the joystick and the locomotion software. The main prerequisite for the system is that the rover is moving according to the requested motions. This feature is formalized by many requirements based on the granularity of the system design and assumptions made over the environment, as listed in Table 1. For example, ϕ_0 describes that the requests sent by joystick are received by the locomotion software, while ϕ_1 describes that the locomotion software receives requests regularly (with a given period of $100ms$).

Fig. 2: The Bridget Rover (courtesy of Airbus Defense and Space UK).

5.2 Nominal Software Design

ID	Label	Formal specification
Requirements on the nominal system		
ϕ_0	Bounded delivery	$\square_{[0,10000]}\ (is\_sent\ \Rightarrow \Diamond_{[0,100]}\ is\_received\_c)$
ϕ_1	Periodic arrival	$\square_{[0,10000]}\ (is\_received\_c\ \Rightarrow \Diamond_{[1,100]}\ is\_received\_c)$
Requirements on the FDIR system		
ϕ_2	Bounded delivery	$\square_{[0,10000]}\ (is\_sent\ \Rightarrow \Diamond_{[0,110]}\ is\_received)$
ϕ_3	Periodic arrival or timeout	$\square_{[0,10000]}\ (is\_received\ \Rightarrow (\Diamond_{[1,110]}\ is\_received)\ \vee\ (\Diamond_{[110,200]}\ is\_timeout))$
ϕ_4	Consistent number of commands	$\square_{[0,10000]}\ (\Diamond_{[0,200]}\ nb\_received = nb\_sent\ +\ nb\_timeout)$
ϕ_5	fault1 detection	$\square_{[0,10000]}\ (cnbt \geq MNBT\ \Rightarrow \Diamond_{[0,200]}\ is\_reset)$
ϕ_6	fault1 repair	$\square_{[0,10000]}\ (is\_reset\ \Rightarrow \Diamond_{[0,100]}\ is\_received)$
Requirements on the system performance		
$\phi_7(n)$	Bounded inconsistency	$\square_{[0,10000]}\ (\Diamond_{[0,200]}\ nb\_timeout - (nb\_received - nb\_sent) \leq n)$
$\phi_8(n)$	Bounded timeouts	$\square_{[0,10000]}\ (nb\_timeout \leq n)$
$\phi_9(n)$	Bounded consecutive timeouts	$\square_{[0,10000]}\ (cnbt \leq n)$
$\phi_{10}(n)$	Bounded Client buffer overflow	$\square_{[0,10000]}\ (nb\_overflow \leq n)$
$\phi_{11}(x)$	Non-deterministic timeouts	$\square_{[0,10000]}\ (nb\_chosen\_timeout \leq x)$
$\phi_{12}(y)$	Non-deterministic *test_cmd* reads	$\square_{[0,10000]}\ (nb\_chosen\_read \leq y)$
$\phi_{13}(n)$	Bounded command offset	$\square_{[0,10000]}\ (\Diamond_{[0,100]}\ period\_id - cmd\_id \leq n)$

Table 1: Requirements of the planetary robotics case study at the different levels of granularity of system design.

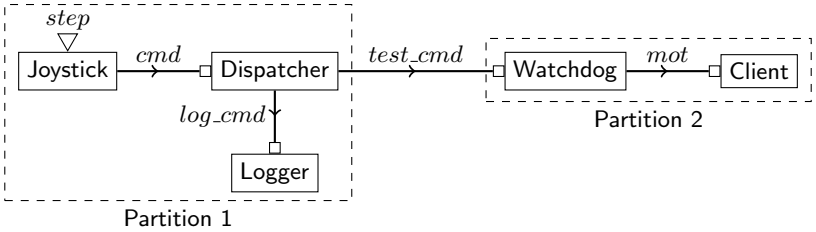


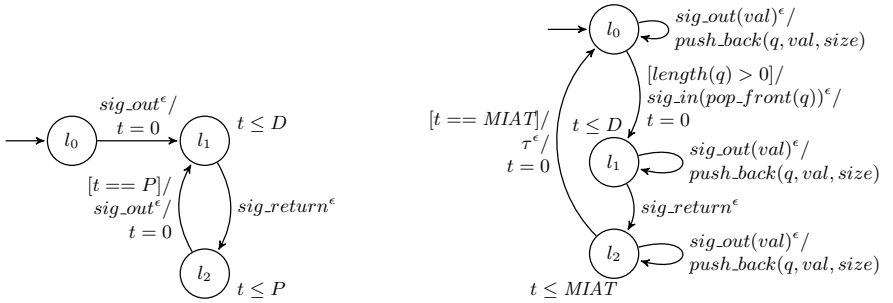
Fig. 3: Overview of the case study software architecture.

Model As mentioned above, the case study tackles the communication of the joystick with the locomotion control system. The software architecture is given is Figure 3.

The software design relies on a library containing two types of components: *triggers* and *queues*. The trigger, illustrated in Figure 4a, is a component that with a customizable period P activates the component to which it is attached (action *sig_out*). Once activated, the trigger waits for the completion of the component's associated behavior (action *sig_return*) before a deadline D. In case the associated behavior does not finish before D, an issue is raised during the analysis. This corresponds a timelock in the system behavior, that is a modeling error and has to be corrected.

The queue, illustrated in Figure 4b, is a component that models the asynchronous communication between two components: a sender and a receiver. It is associated with the receiver component and it contains a buffering structure of fixed *size* to store received requests (on action *sig_out*). If the limit has been reached, the incoming requests are discarded. When available, the receiver consumes the requests stored in its queue, using action *sig_in*, with a fixed minimal time between two reads, called *MIAT* (minimal inter-arrival time). Similarly to the trigger, the queue waits for the completion of the receiver's associated behavior (action *sig_return*), that must happen before a deadline D.

The Joystick component regularly sends a motion command denoted *cmd* to the rover locomotion software to be executed. This behavior is depicted in Figure 5a by the states l_0 to l_4, represented in black. The command sending is activated by the *step* trigger. For this case study, the *step* period and deadline are set to $100ms$ and $15ms$ respectively. The *cmd* request has multiple fields to describe the motion to be executed: an *id* of type integer records the package number, and *motion* records the actual data package sent to the locomotion software consisting of translation, rotation and heading of type float.



(a) Generic behavior of triggers with parameters P and D

(b) Generic behavior of queues with parameters $MIAT$, D and $size$

Fig. 4: Library of components and their behavior: triggers represented with triangle ($\bigtriangledown$) and queues represented with square ($\square$) in Fig. 3.

The *cmd* action is first sent to a Dispatcher, provided in Figure 5b. The Dispatcher transfers this request to two software components: first the Logger via the *log_cmd* request and then to the Watchdog via the *test_cmd* request (states l_0, l_3, l_4 and l_6 in black). Both *log_cmd* and *test_cmd* contain the data package received from the Joystick. The Logger records the received requests such that they can be reused later for the validation of the system through replaying.

The Watchdog interfaces the Dispatcher (and subsequently the Joystick) with a wrapper of the locomotion control software called Client, as illustrated in Figure 5c by the states l_0 to l_3 in black. Whenever a *test_cmd* request is received, the Watchdog transfers the data package in the *mot* request to the Client. For simplicity, we do not consider the behavior of the Client, which is abstracted to discarding all received *mot* requests.

All these components communicate asynchronously via queues. All queues have the $MIAT$ and D set to $50ms$ and $15ms$ respectively, while they can record only one element, i.e., $size = 1$.

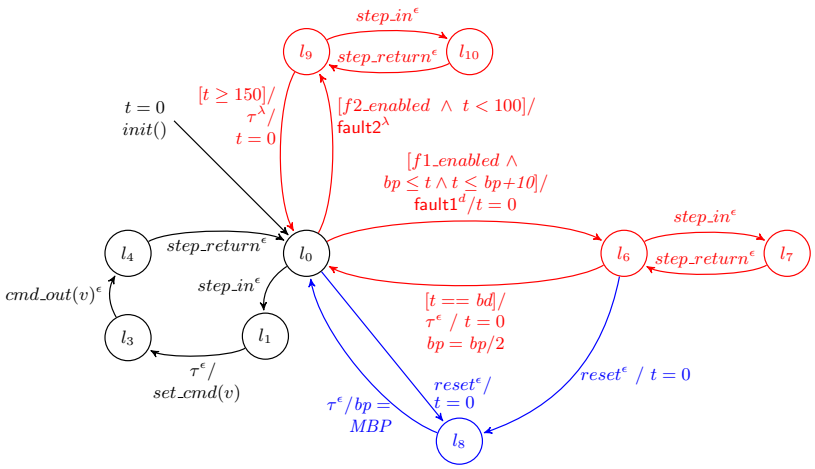
Validation Requirements For this case study we are interested first in validating the nominal behavior of the system when the environment assumptions are satisfied:

(A1) the Joystick issues periodically a *cmd* request,
(A2) no requests (data packages) are lost, and
(A3) all executions including queue writing and reading take $0ms$.

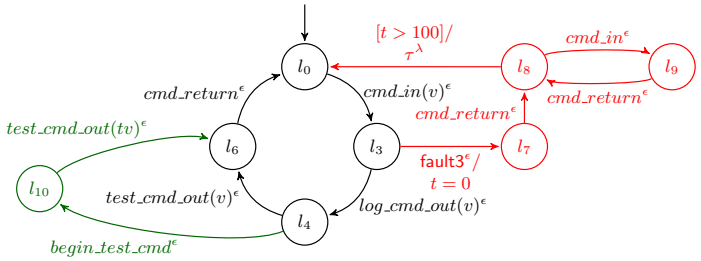
The system's prerequisite – the locomotion system executes the received motion commands – is expressed on the nominal application by the requirements ϕ_0 and ϕ_1. ϕ_0 describes that all *cmd* requests sent by the Joystick, modeled with the Boolean variable *is_sent*, are received within $100ms$ by the locomotion system (Client component), modeled with the Boolean variable *is_received_c*. ϕ_1 describes that the Client regularly receives a request *mot*, within $100ms$. Please note that these requirements and their formalization are relaxed with respect to assumption (A3) as they describe cyclic behavior within periods.

Validation Results We use the SMC-BIP tool with the confidence parameters $\alpha = 0.005$ and $\delta = 0.05$ for all our experiments (at this step and after). These confidence parameters require the evaluation of 1199 system executions to come up with a global verdict, using the probability estimation technique. The computed satisfaction probability is 1 for both ϕ_0 and ϕ_1. The computations took for each requirement independently roughly 4 min.

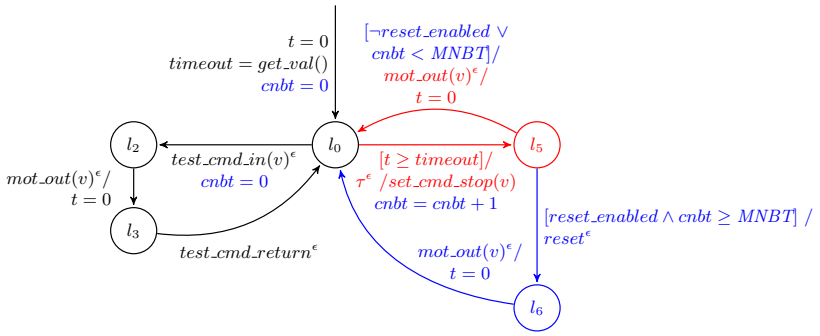
Conclusion As the requirements on the nominal application are satisfied, we can proceed with the next step of the approach.



(a) Behavior of Joystick



(b) Behavior of Dispatcher



(c) Behavior of Watchdog

Fig. 5: Behavior of the main components from Fig. 3 represented as timed automata in $\mathcal{S}$BIP, where faults, fault detection and standard recovery action are represented in red, more complex recovery strategy in blue, and deployment-specific actions in dark green.

6 Risk Assessment of the Planetary Robotics System

In the following we describe two models at different levels of granularity and the results of the risk assessment and validation phases on them: one system level design including faults modeling relaxed assumptions and one deployement level design additionally modeling the hardware platform.

6.1 On Robustness to Faults

Model with Faults The first transformation of the nominal model ($i = 1$) tackles the assumptions (A1) and (A2) described in Section 5.2. We relax these assumptions in the new model by incorporating faults. In order to have a systematic way of evaluating the impact of faults and their combinations on the system and its requirements, we define Boolean constants for each fault to control their injection.

Assumption (A1) describes that the Joystick sends periodically a request. We break this hypothesis by modeling the stop of request sending for a certain amount of time with two faults as follows. The fault1 action is related to the external code that can be embedded in the model. More specifically in this case, the software of the Joystick component does not send *cmd* for a certain duration. This behavior is represented in Figure 5a by the red states l_6 and l_7, as follows. The fault is injected at state l_0 with the Boolean constant $f1\_enabled$ and can be executed any time between bp and $bp + 10$. (bp is an integer variable that describes the break period of not sending *cmd* requests.) Once the transition is picked for execution, in a uniform way, the clock t that measures the break duration bd is initialized to 0. During the $[0, bd[$ time interval, any step triggers are discarded by the Joystick as modeled with the $l_6 - l_7$ cycle. Once the break duration bd has passed, the Joystick recovers, the clock is reset and the break period is updated. For this example, bd is equal to $20ms$ and bp is set to $190ms$. Please note that we model here a persistent fault since the break period is decreasingly converging to $0ms$, and therefore the Joystick will eventually be continuously failing.

The fault2 action is motivated by the risks in the hardware connections, where the Joystick can be unplugged non-deterministically for a certain moment. This behavior is modeled in Figure 5a by the red states l_9 and l_{10}. As before, this fault is injected at state l_0 by the Boolean variable $f2\_enabled$ and can be executed as long as $100ms$ have not passed since the last clock reset. If the fault occurs, the Joystick could recover after $150ms$ since last time reset (the transition from l_9 to l_0). However, the recover action is defined as *lazy*, which implies that the Joystick could fail for large time periods. During the fail, any step triggers are discarded as above.

Assumption (A2) describes that no motion command is lost. We relax this hypothesis in the Dispatcher, where we model the message loss with the fault3 action. This behavior is represented in Figure 5b by the red states l_7, l_8, and l_9. As above we inject the fault with the Boolean constant $f3\_enabled$ and we consider that this fault can happen after a *cmd* is received. Then the Dispatcher has the choice of forwarding the data package (transition between l_3 and l_4) or losing the data package (transition l_3 to l_7). If a *cmd* is lost, the Dispatcher will continue to loose packages (cycle $l_8 - l_9$) unless it recovers. The recovery cannot happen before $100ms$ since fault3. Again, the recovery is *lazy* (transition from l_8 to l_0), which means that the Dispatcher can loose commands for large time periods.

Risk Assessment Requirements For the risk assessment we use the requirements ϕ_0 and ϕ_1 defined above, and we do not introduce other ones. These requirements are sufficient to quantify and assess the impact all faults – fault1, fault2, and fault3 – have on the nominal behavior on the system.

Risk Assessment Results With the SMC-BIP tool we obtain the following resuls. Regarding fault1 and fault2, ϕ_0 is satisfied with probability 1. Indeed, if a command is not sent the implication evaluates to *true*. However, this probability drops to 0 when injecting fault3. In this case, some sent commands are lost by the Dispatcher. Therefore, they are not received by the Client. Property ϕ_1 is not satisfied regardless of the faults occurring, independently or combined, since commands are either not sent or lost within the $[1, 100]ms$ time interval. The computed probability is equal to 0. The complete results, including the time needed for the experiments, are given in Table 2.

	fault1 (without reset)		fault1 (with reset)		fault2		fault3		fault4 ($MTD = 0/MTD = 5$)	
	Probability / Parameter	Time (sec)	Probability / Parameter	Time (sec)	Probability / Parameter	Time (sec)	Probability / Parameter	Time (sec)	Probability / Parameter	Time (sec)
Nominal system										
ϕ_0	1	240	1	196	1	229	0	14	-	-
ϕ_1	0	14	0	13	0	14	0	14	-	-
System with FDIR functionality										
ϕ_2	1	240	1	196	1	229	0	14	0.05 / 0	14 / 96
ϕ_3	1	202	1	242	1	211	1	180	1	300
ϕ_4	1	192	1	246	1	215	0	14	0.05 / 0	14 / 105
ϕ_5	-	-	1	194	-	-	-	-	-	-
ϕ_6	-	-	1	217	-	-	-	-	-	-
$\phi_7(n)$	$n^*_{\phi_7(n)}$=0	140	$n^*_{\phi_7(n)}$=0	140	$n^*_{\phi_7(n)}$=0	140	$n^*_{\phi_7(n)}$=88	900	$n^*_{\phi_7(n)}$=4	2160
$\phi_8(n)$	$n^*_{\phi_8(n)}$=88	720	$n^*_{\phi_8(n)}$=61	197	$n^*_{\phi_8(n)}$=88	720	$n^*_{\phi_8(n)}$=88	900	$n^*_{\phi_8(n)}$=49	2160
$\phi_9(n)$	$n^*_{\phi_9(n)}$=88	720	$n^*_{\phi_9(n)}$=5	180	$n^*_{\phi_9(n)}$=35	1800	$n^*_{\phi_9(n)}$=30	1080	$n^*_{\phi_9(n)}$=1	480

Table 2: Results obtained with the $\mathcal{S}$BIP framework on the system design with faults and with respect to requirements from Table 1. n^*_ϕ refers to the parameter value for which $\phi(n)$ is satisfied with probability 1.

Given these results, we conclude that these faults (i.e., risks) have a great impact on the system and an FDIR behavior needs to be added such that the rover operates safely. Indeed, without a recovery strategy, the rover's locomotion system would keep executing the last received command and this can have important consequences. For example, the rover could be stuck into a harmful environment and therefore not achieve its mission. More specifically, we are interested in stopping the rover whenever such faults are detected.

Model with FDIR Behavior In order to stop the rover consistently when risks are present, we equip the Watchdog with a data package validity checking before transferring the request to the Client. By validity we do not mean the checking of the command content (even though this could be achieved if necessary), but ensuring that *the package must be received before a* timeout *event*. This corresponds to the diagnoser part of FDIR components, and it is based on the property that the Client awaits periodically for motion requests.

If this does not happen due to faults in the system, the Watchdog will ensure the rover still operates safely with respect to the locomotion system: *the motion is stopped*. To achieve this, the Watchdog sets the data package to stop – translation, rotation and heading are set to 0 – and sends *mot* with stop as value. This corresponds to the controller part of FDIR components.

For simplicity[2], we model this FDIR behavior directly in the Watchdog as illustrated in Figure 5c. While waiting for a *test_cmd*, the Watchdog checks the time elapse with the clock t. If the timeout duration has been observed (transition from l_0 to l_5 in red), the stop data package is set, the corresponding *mot* command is sent and the clock is reset (transition from l_5 to l_0 in red). We configure the value of the timeout to $110ms$ in order to account for possible delays in the system execution. As a consequence, the requirements listed in Table 1 for the FDIR component and described below use this new time interval in their formalization.

We can now proceed with validating the FDIR behavior of the Watchdog component.

Validation Requirements We define a new set of requirements ϕ_{2-4} specific to the FDIR behavior of the Watchdog. ϕ_2 describes that whenever a motion command is sent, it is received by the Watchdog. This requirement is the transformation of ϕ_0 from the Client to the Watchdog as receiver. Indeed, there is no need to check ϕ_1 from now on, as the new FDIR behavior should guarantee it. With ϕ_3 we are interested to check that the Client should receive *mot* requests periodically. The *mot* could contain either the data

[2] The system architecture and specification, Watchdog included, have been provided in the frame of this case study such that the used resources (e.g., number of components and threads) are minimal.

package sent by the Joystick (modeled with the *is_received* variable) or the stop data package sent by the Watchdog (modeled with the *is_timeout* variable). Note that ϕ_3 is also a transformation of ϕ_1 from the Client to the Watchdog by including the FDIR part. Requirement ϕ_4 models the consistency of the data package reception: all the data packages received by the Client are either commands generated by the Joystick or by the Watchdog.

Validation Results The Watchdog is robust with respect to faults fault1, fault2, and fault3: the probabilities computed for ϕ_3 are 1, as showed in Table 2. In addition, ϕ_2 and ϕ_4 are also satisfied when considering the faults of the Joystick, namely fault1 and fault2. However, in the presence of fault3 of the Dispatcher, the probability of satisfaction for ϕ_2 and ϕ_4 is 0. This result is expected since fault3 models message loss. Whichever combination of faults between the two components is considered, the results are similar: ϕ_3 is satisfied, while ϕ_2 and ϕ_4 are not.

Conclusion This step of the approach consisted of the following actions. The system model was enriched with faults and the need of FDIR behavior was highlighted by the risk analysis and evaluation results. The decision was to include the detection and recovery capabilities into the Watchdog component which now also checks the validity of the received commands. We showed that the Watchdog is robust with respect to the modeled faults. As the obtained results are satisfying, we go to the next step $i = 2$ in our approach.

6.2 On System Performance

Model for Performance Measurement. At this step we are interested in the performance of the FDIR behavior of the Watchdog in the system. Therefore we do not perform any model transformation – ${\Gamma_2^1}^{(0)}$ is the identity function. However, we enrich the set of requirements with ones evaluating the system performance ϕ_{7-9}, as described below.

Risk Assessment Requirements. ϕ_7 explores the different bounds for inconsistency in the number of packages. This requirement makes sense to be checked when the system looses commands, i.e., when fault3 is present or $P(\phi_4) \neq 1$. ϕ_8 explores the maximal number of stop commands the Watchdog issues for the given time period, while ϕ_9 considers the number of consecutive stop commands. We are interested in this case to have a low number of stop commands such that the rover operates smoothly.

Risk Assessment Results. The results obtained from the risk assessment are given in Table 2. We remark that fault1 leads the Watchdog to issue a large number of (consecutive) stop commands ($\phi_{8,9}$) due to its persistence. As we will see in the next refinement, we tackle this aspect by introducing a reset mechanism. fault2 and fault3 imply the same large number of stop commands from the Watchdog as fault1. However, we remark that in these cases, the system recovers for longer periods and acts consistently since the consecutive number of stop commands is bounded to 35 and 30, respectively, as illustrated in Figure 6 and 7. The maximal failure period is approximately equal to $35 \times 100ms$ and $30 \times 100ms$, respectively. Therefore, the Watchdog is able to keep the system safe, even with long failure periods.

Model with *reset* Mechanism for the Joystick Since fault1 is persistent and the rover will be mostly not moving due to the stop commands issued by the Watchdog, we add a reset mechanism in the Joystick that will allow this component to go back to its nominal behavior. This mechanism, illustrated in Figure 5a in blue, consists of an action *reset* (leading to state l_8) which sets the break period *bp* back to the maximal allowed duration *MBP* (transition from l_8 to l_0). Then the Joystick will again issue motion commands, with fault1 enabled.

The reset mechanism is controlled by the Watchdog since it implements FDIR behavior and it is enabled with a Boolean constant *reset_enabled* (as for fault injection). As illustrated in Figure 5c (in blue), the

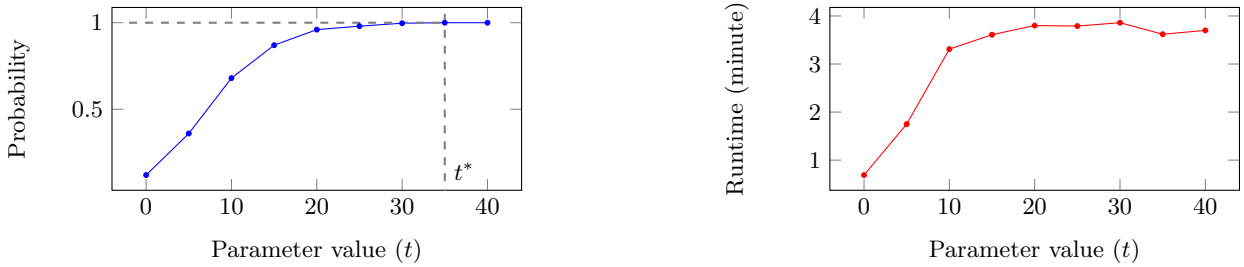


Fig. 6: Probability and runtime of ϕ_9 for the model including fault2.

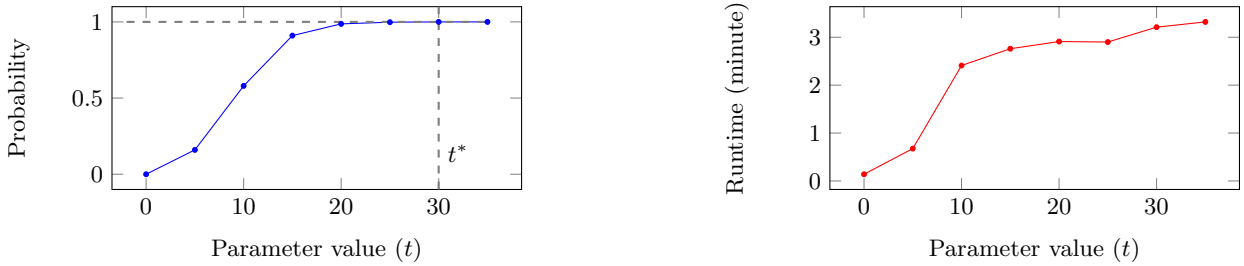


Fig. 7: Probability and runtime of ϕ_9 for the model including fault3.

Watchdog defines a variable *cnbt* that stores the consecutive number of stop commands issued. If *cnbt* is below the *MNBT* threshold, the Watchdog will issue a stop command (transition from l_5 to l_0). Otherwise, the Watchdog will first trigger the reset mechanism (transition from l_5 to l_6) and then will issue the stop command (transition from l_6 to l_0). Please note that the behavior of the watchdog described in Section 6.1 is identical to the one described in this figure, when *reset_enabled* is set to *false*.

Validation Requirements The efficiency of the reset mechanism is additionally validated by requirements ϕ_5 and ϕ_6. ϕ_5 describes that whenever fault1 is detected, the Watchdog triggers the *reset* action. ϕ_6 validates the efficiency of the reset mechanism modeled by the receiving of a command by the Watchdog after a reset is triggered. We also check and compare the performance of the reset mechanism with requirements ϕ_{7-9} described above.

Validation Results For this model, we configure the Watchdog to tolerate a maximum number of 5 consecutive timeouts before triggering a Joystick reset ($MNBT = 5$).

In the second column of Table 2, we see that both ϕ_5 and ϕ_6 are satisfied with probability 1. From the performance point of view, we observe an improvement of order of magnitude for the number of stop commands issued by the Watchdog. More specifically, the total number of issued stop commands is reduced by 31%, whereas the number of consecutive stop commands is bounded to 5. The latter corresponds in general to the bound *MNBT* for which the reset mechanism is implemented, instead of the computed bound of 88 without the reset mechanism.

An interesting result is obtained when combining fault1 implementing the reset mechanism and fault3. In this case, requirement ϕ_6 does not hold. The reason is that the reset mechanism does not guarantee the receiving of commands by the Watchdog at the next cycle. This is mainly due to the fact that the consecutive timeouts could be caused by the Dispatcher and in that case, resetting the Joystick will not change anything.

Another refinement is necessary in order to develop a more resilient system by implementing more complex recovery strategies.

Conclusion At this step we checked the performance measurements of Watchdog in the system. We observe that in some cases the Watchdog is efficient. In others, as for example fault1, a more complex recovery mechanism based on the *reset* of the Joystick is implemented and validated. It is showed that the robustness property (ϕ_3) is preserved by the model with the reset mechanism, and moreover this mechanism reduces the overhead of the Watchdog on the system performance. Therefore, we proceed by transforming and studying a deployment model of our system.

6.3 On Deployment Impact

Deployed Model The software is deployed on two partitions due to the rover architecture. The locomotion control software, and therefore its Client wrapper, are deployed on a partition which communicates with the other software components via a CAN-bus. Since the Watchdog component aims to check the validity of the requests sent to the Client, it will be deployed on the same partition with the Client – Partition2 in Figure 3. The remaining components – Joystick, Dispatcher, and Logger – are deployed on another partition – Partition1 in Figure 3. Between the two partitions a Channel component is considered.

The Channel is a component added to the system model and connected to the Dispatcher in writing mode and to the Watchdog in reading mode. This component, illustrated in Figure 8, has a similar behavior to the queues. Once a data package is received, it is written in a buffer of a predefined *size*. If the Channel buffer is full, the received data package is discarded. Finally, the recorded data package is removed from the buffer and transferred to its target when possible.

We relax here assumption (A3) and we model *maximal transmission* and *writing delays* for the Channel with variables *MTD* and *MWD*, respectively. The maximal transmission delay describes how much time a data package transfer can take at most. For example, such a variation in the transmission time can depend on the network load. This behavior is modeled by the transition from l_1 to l_2 in Figure 8: a transfer can finish at any moment between 0 and *MTD* (the delayable d urgency). Similarly, the maximal writing delay describes the time the Dispatcher can take to write a motion command in the Channel buffer before continuing its behavior (here, informing the Dispatcher queue that another request can be handled). Usually writing on a channel is not instantaneous. We consider that the writing can take between 0 and *MWD* as described by the guard on transition from l_0 to l_1 and loop transitions in states l_1 and l_2. The writing process is initialized by the Dispatcher with the *begin_sig_out* action. This action can always be performed on the Channel, as modeled by the self-loop transitions labeled with *begin_sig_out* in all states (Figure 8).

Moreover, we enable the loss of command requests with fault4. More precisely, a package stored in the buffer (state l_1) can be lost at any moment (the lazy λ urgency). Similarly to the other faults, we use the Boolean constant *f4_enabled* to inject it. This fault implies the removal of the oldest data package from the buffer. This can result in an empty buffer (transition from l_1 to l_0) or a buffer with at least one data package (loop transition in l_1).

In this setting, fault1 to fault3 are disabled. These faults can be considered in a further step together with the current deployment model.

Risk Assessment Requirements We consider for evaluation requirements ϕ_{2-13} from Table 1. Please note that ϕ_5 and ϕ_6 are not checked since they make sense only when fault1 is present. ϕ_{10} explores the number of *mot* requests lost by the Client queue: incoming requests are lost if the queue is full. ϕ_{11-13} tackle the freshness of the data package received by the Client. We want to determine with ϕ_{11} how many times the Watchdog triggers a stop command when a motion command is present in its queue. ϕ_{12} is the dual of ϕ_{11}: how many times the Watchdog handles a command received at timeout. Finally, ϕ_{13} explores the discrete time difference (in terms of periods) between when a command is received and when it is issued.

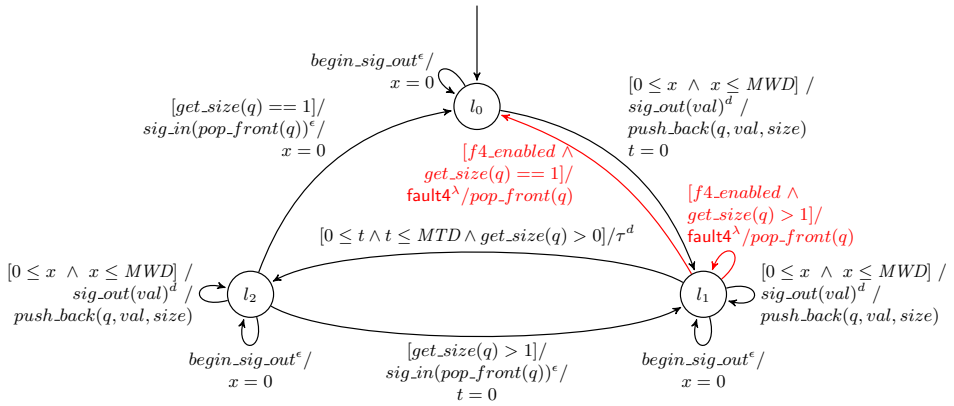


Fig. 8: Behavior of the communication channel between the two partitions of Fig. 3.

Risk Assessment Results For this analysis, we consider 3 aspects of the Channel in the deployed model: (1) $M_3^{1(0)}$ with transmission delays, (2) $M_3^{2(0)}$ with writing delays, and (3) $M_3^{3(0)}$ with command losses.

Transmission delays. This exploration concerns the *MTD* parameter of the model, with $MWD = 0$. The results obtained with SMC-BIP for this model on requirements $\phi_{2-4,7-10}$ are represented in Figure 9. The evolution of the probability estimation for ϕ_{2-4} and $\phi_{10}(0)$ is plotted on the left, while the evolution of the optimal parameter value for ϕ_{7-10} is displayed on the right. Notice that both the probabilities and the optimal parameter values are functions of the maximal transmission delay (*MTD*) represented on the x-axis, and which is a parameter of the model as described above. The optimal parameter value for a property ϕ is the smallest parameter value for which the property is satisfied with probability 1. We write the optimal parameter as $n^*_\phi = min_{n \in \mathbb{N}}\{\mathcal{P}(\phi(n)) = 1\}$.

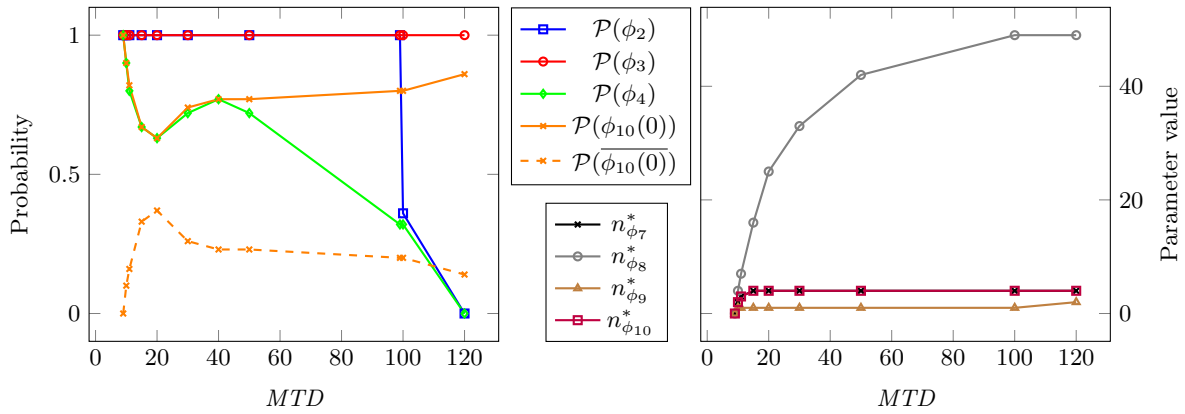


Fig. 9: SMC-BIP results for the deployed model including transmission delays.

We start by presenting the parameter estimation results on requirements ϕ_{7-10}. The total number of stop commands as described by ϕ_8 is in general bounded to 49. The number of consecutive stop commands, as expressed by ϕ_9, is bounded to 1 for *MTD* values below $100ms$. Similarly, the number of lost commands due to the Client queue overflow modeled in ϕ_{10} is bounded to 4 motion commands. We remark that $n^*_{\phi_7} = n^*_{\phi_{10}}$, which shows that the Client queue overflow is the only source of command losses in the system.

The Client queue can discard a message due to being full if all of the following three conditions are satisfied in the given order: (i) a *test_cmd* arrives at exactly timeout for the Watchdog, (ii) the Watchdog chooses to first issue a stop command, (iii) the Watchdog immediately transfers the *test_cmd* as *mot*, without the Client handling the previous stop data package. Then, the queue of the Client which size is 1 will not be able to store the *mot* and this data package is dropped. This situation evolves with respect to *MTD* as follows. When *MTD* increases, the number of stop commands issued also increases as showed for ϕ_{11} up to a certain limit. However, the probability for the *test_cmd* sent by the Dispatcher to take exactly timeout $- P$ to be delivered to the Watchdog is decreasing (i.e., it boils down to generating a single value in the growing interval $I = [0, MTD]$ of transmission delays).

Therefore, we observe that the probability to loose commands, represented by the negation of $\phi_{10}(0)$ on the left hand side of Figure 9, first increases until $MTD < 20ms$ and then decreases. The increase is justified by the higher impact the choice of sending stop commands instead of transferring the motion requests (see the results for ϕ_{11} in Figure 10) has on the property. After $20ms$, the aforementioned choice stabilizes and the probability to generate a single time value in I decreases, leading $\mathcal{P}(\overline{\phi_{10}(0)})$ to also decrease.

Finally, on the left hand side plot of Figure 9, we observe that ϕ_2 is satisfied when *MTD* is lower than $100ms$. However, in the cases where the transmission delay is greater than the $100ms$ period of the Joystick, the delivery of commands is no longer guaranteed in the same period. It is worth mentioning that the Watchdog is able to detect that phenomenon and act accordingly, as reflected by requirement ϕ_3 that is satisfied with probability 1. With respect to requirement ϕ_4, we remark that the interleaving of command generation and reception also can have an effect on the probability of satisfaction, besides the command loss. The interleaving can happen when the *MTD* is greater than $40ms$ (as expected from the timeout value) and therefore the satisfaction probability of ϕ_4 decreases. Up to $40ms$, ϕ_4 and $\phi_{10}(0)$ are identical, while after $40ms$ they diverge.

Next, we are interested in quantifying the number of times the Watchdog has the non-deterministic choice between issuing a stop command (ϕ_{11}) or reading a command from its queue (ϕ_{12}). These results are showed in Figure 10. We observe that the number of non-deterministic choices, represented by the sum of x^* and y^* values, varies with the *MTD* then stabilizes at the value of 6 when the transmission delay is above $20ms$. Also, the Watchdog chooses fairly between the two options. An interesting observation is that the number of chosen stop commands is relatively low in comparison with the total number of issued stop commands, as reflected in Table 3.

Fig. 10: Parameter exploration of ϕ_{11} (left) and ϕ_{12} (right) on the deployed model with transmission delays

Lastly, we focus on analyzing motion command and period identifiers to evaluate data freshness. Due to transmission delays, a command can be received in a different period than the one it was issued in. We express this behavior with ϕ_{13} and the results are showed in Figure 11. The left plot represents the variation of the probability estimation of $\phi_{13}(n)$ for different instances of n when varying the *MTD*. The right plot represents the optimal value of n for different values of *MTD*. We can see that the difference between the period of issued and received command identifiers is bounded and increases with the growth of *MTD*. Hence the transmission delays have an important impact on data freshness.

MTD	Max_timeouts ($n^*_{\phi_8}$)	#choose_timeout (x^*)	Proportion $x^*/n^*_{\phi_8}$
10	4	2	0.5
20	25	3	0.12
30	33	3	0.09
80	45	3	0.07

Table 3: Proportion of non-deterministic stop commands when increasing MTD.

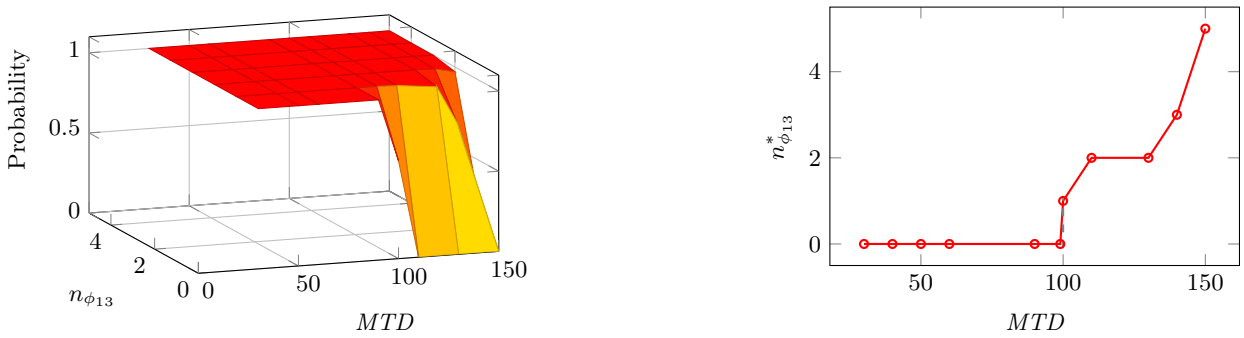


Fig. 11: Parameter exploration of ϕ_{13} on the deployed model with transmission delays.

In order to deal with lost commands in the Client queue, a refinement could be performed by increasing the size of this queue. A binary search can be used to determine the minimal size. In our case, a queue of size 2 is sufficient to avoid overflows, as shown in Figure 12 for acceptable ($30ms$) and high ($100ms$) values of MTD. The probability to not loose any commands in the Client's queue ($\phi_{10}(0)$) is equal to 1. In the case where $MTD = 100ms$, the difference in the number of commands received by the Client and those sent by the Joystick and Watchdog expressed by ϕ_4 is only due to the interleaving of actions.

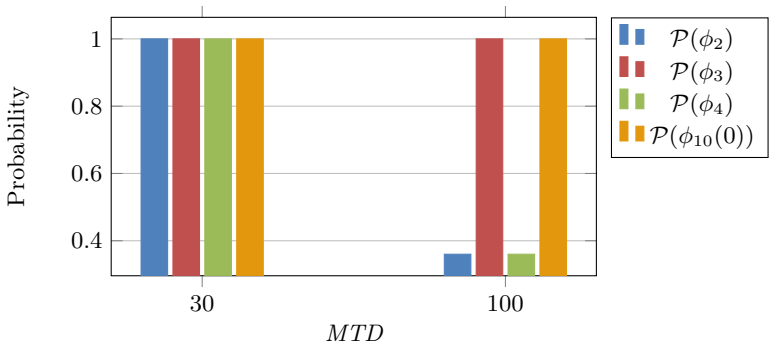


Fig. 12: Results on the corrected deployed model.

Writing delays. This exploration concerns the MWD parameter of the model, with $MTD = 0$. Recall that the D (deadline) of the Dispatcher queue is set to $15ms$. Indeed, the Dispatcher has to transfer the *cmd* request as *log_cmd* (which takes $0ms$) and as *test_cmd* via the Channel (which takes at most $MWDms$). All these actions must happen before the D implemented by the queue, otherwise the Dispatcher timelocks. Timelocks are modeling errors that can be detected during model analysis and can be subject to model transformation:

for example, either set a new worst case execution time for the component and in consequence the system scheduling is recomputed, or a recovery mechanism (similar to the reset mechanism proposed) is implemented in the Dispatcher.

The results are provided in Figure 13. In the left hand side plot we illustrate the probabilities estimated for ϕ_{2-4}. When $MWD < 15ms$, we observe that the Watchdog is robust (ϕ_3). Additionally, analysis results for $\phi_{2,4}$ in this setting are similar to the results in the transmission delay setting. When $MWD \geq 15ms$, most requirements do not hold anymore due to the timelock of the Dispatcher. However, the Watchdog keeps on guaranteeing the system's safety, as shown by the probability of ϕ_3 evaluated to 1. Note that $\phi_{10}(0)$ is equivalent to ϕ_4 because there is no interleaving possible between the command generation and reception when the $MWD < 40ms$.

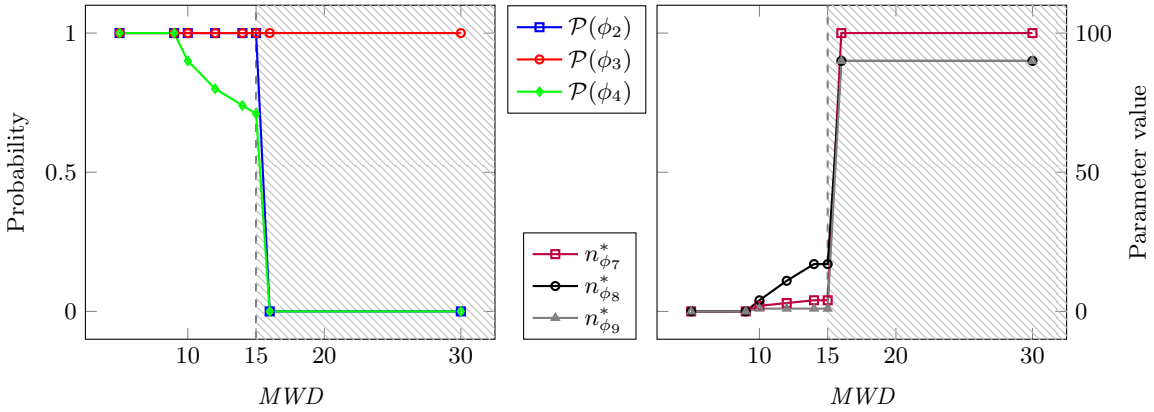


Fig. 13: SMC results for the deployed model with writing delays

On the right hand side we illustrate the optimal parameter values obtained on properties ϕ_{7-9}. These results are similar to those obtained in the transmission delay setting for $MWD < 15ms$. Note that in this case $\phi_{10}(n)$ is identical to $\phi_7(n)$, and therefore it is not explored.

Command Losses. Finally, we consider the command loss aspect of the Channel. For our exploration we set $MTD \in \{0ms, 5ms\}$ and $MWD = 0ms$. The results with respect to $\phi_{2-4,8-10}$ are given in Figure 14. The results for requirements ϕ_{11-13} are not relevant in this case since MTD should be greater than $10ms$ and $40ms$ for a relevant exploration of ϕ_{11-12} and ϕ_{13}, respectively.

We remark that ϕ_3 is always satisfied. Requirements $\phi_{2,4}$ reflect the occurrence of commands losses: the satisfaction probability is 0.05 when $MTD = 0ms$, and 0 when $MTD = 5ms$. Property $\phi_{10}(0)$ is satisfied with both values of the transmission delay. This shows that commands are not lost in the client's queue due to an overflow; the only source of command losses in this model is fault4. Consequently, the number of timeouts encodes the number of times the fault occurred. When there is no transmission delay, the number of lost commands is bounded to a value of 8 ($n^*_{\phi_8}$) with a maximum of 3 consecutive losses ($n^*_{\phi_9}$), as shown in Figure 14 for $MTD = 0ms$. These numbers drastically increase when $MTD > 0ms$. For instance when $MTD = 5ms$ illustrated in Figure 14, the Channel can loose up to 25 commands ($n^*_{\phi_8}$) with a maximum of 5 consecutive losses ($n^*_{\phi_9}$).

Conclusion We have modeled a deployment for our case study taking into account three risk cases: transmission delays, writing delays and command losses. For each of these aspects we have explored ϕ_{2-13} to evaluate the impact of these risks on the behavior of the rover. Based on the shown results, 2 scenarios are further possible depending on other high level requirements. If the hardware constraints obtained through

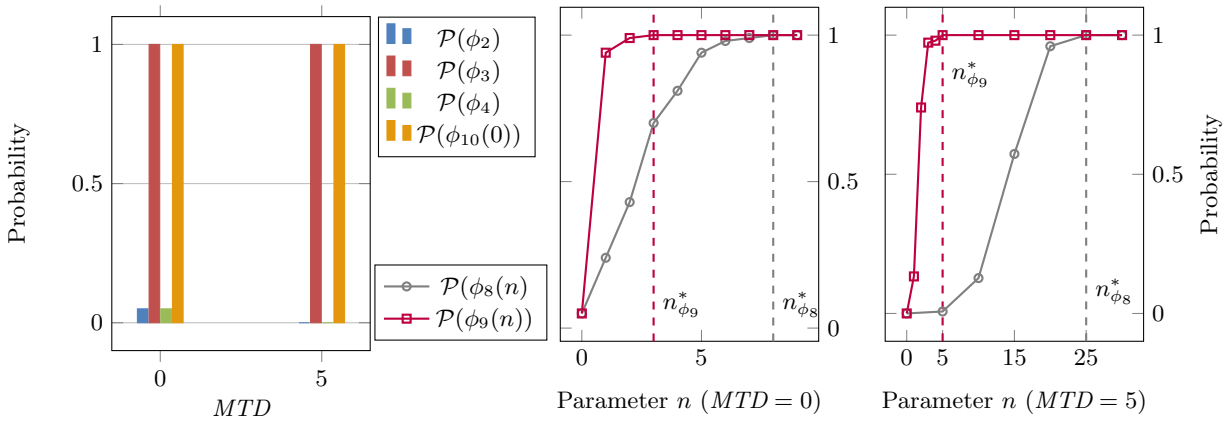


Fig. 14: SMC results for the deployed model with command losses

exploration are implementable, the designer can continue with the deployment. Otherwise, a model transformation and a new exploration are required to correct the undesired behaviors and identify the needed hardware for the deployment, as briefly described for the transmission delay risk above.

In this design, we note that whatever the risk is, the Watchdog is robust and the system's safety with respect to the motion functionality is guaranteed. This is also true for the timelock modeling error present in the Dispatcher that occurs for writing delays starting from $15ms$. Before $15ms$, the transmission and writing delays have a similar impact on the system behavior. For higher values, however, these risks have an impact on different aspects of the system. Transmission delays can imply the non-satisfaction of data freshness, meaning that commands are no longer received in the $100ms$ period since their generation. Writing delays can lead to command losses and the generation of numerous stop data packages. In the latter case, the system enters a degraded mode where generated data packages are no longer delivered.

A shared effect of these risks on the locomotion system is the overflow in the Client queue. To address this, a refinement can be necessary, such as the one proposed above where the Client queue size is increased to 2. Therefore, we can safely state that the model does not loose any command and the timelock in the Dispatcher is avoided as long as $MWD < 15ms$. Additionally, if $MTD < 100ms$, the data freshness property is satisfied. Given the deployed system model and the explorations we performed, the values obtained here are optimal in the sense that they guarantee all requirements desired for the system.

7 Discussion

In this paper, we propose a model-based design approach that relies on formal methods to develop real-time resilient systems. The method is incremental: it starts from the nominal model, then transformations are applied to take into account different sources of risks. The impact of the considered risks is evaluated using a quantitative risk assessment method and FDIR components are introduced accordingly. These are then validated against safety and performance properties. The approach was successfully used for the design and validation of the control software of a planetary rover.

Approach. Following a model-based approach for the design of FDIR components and their validation provides a lot of flexibility and allows to explore various situations rapidly. Combined with formal methods, it provides more confidence in the obtained results given that the built models are faithful, which is not trivial and requires some expertise. Finally, the use of statistical model checking automates quantitative risk analysis, and helps to deal with real-life system models. However, both the identification and the evaluation of risks remain manual and subject to the designer's interpretation.

Case study. The results presented in the paper are part of the work realized for the validation of the ESROCOS environment [32] with a real-life robotics case study. Although the approach was successfully applied and the designed system is currently being tested in field trials, we wish to share some of the challenges we faced. Building faithful models is by far the most challenging. The choice of the appropriate abstractions to perform and the probability distributions to use requires a deep knowledge of the system under analysis. Using risk assessment helped to take well founded decisions in order to build robust FDIR components. However, the notion of risk is large and several times we found ourselves analyzing risks at different levels, such as risks due to faults then risks due to adding new FDIR behavior, etc. Moreover, managing the transformed models and the associated requirements can quickly become cumbersome if not methodically performed.

Tools. Risk analysis automation is primordial for the design of complex systems as the design space is substantial and proceeding manually is not feasible. In our case, once we built a model it becomes almost straightforward to analyze it using the SMC-BIP engine. Nevertheless, some difficulties remain to use the tool properly, like the correct formalization of requirements in MTL or the instrumentation of the model in order to perform SMC.

Future work. In this paper, we only considered quantitative risk assessment. Using qualitative assessment before may help a lot in filtering irrelevant risks with respect to the requirements of interest. Moreover, risk identification could be done in a knowledge-based manner by using machine-learning techniques for instance. Finally, we are also interested to evaluate the applicability of the approach to security risk assessment. Indeed, we believe that our approach is general enough to also handle security requirements.

References

1. ESROCOS Planetary Exploration Demonstrator. https://github.com/ESROCOS/plex-demonstrator-record
2. ESROCOS Project Github Repository. https://github.com/ESROCOS
3. Abdellatif, T., Bensalem, S., Combaz, J., de Silva, L., Ingrand, F.: Rigorous design of robot software: A formal component-based approach. Robotics and Autonomous Systems 60(12), 1563–1578 (2012), https://doi.org/10.1016/j.robot.2012.09.005
4. Alur, R., Henzinger, T.: Real-time logics: Complexity and expressiveness. Information and Computation 104(1), 35 – 77 (1993)
5. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theor. Comput. Sci. 126(2), 183–235 (Apr 1994)
6. Ashibani, Y., Mahmoud, Q.H.: Cyber physical systems security: Analysis, challenges and solutions. Computers & Security 68, 81–97 (2017)
7. Baier, C., Katoen, J.P.: Principles of Model Checking (Representation and Mind Series). The MIT Press (2008)
8. Basu, A., Bensalem, S., Bozga, M., Caillaud, B., Delahaye, B., Legay, A.: Statistical Abstraction and Model-Checking of Large Heterogeneous Systems. In: Forum for fundamental research on theory, FORTE'10. LNCS, vol. 6117, pp. 32–46. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
9. Basu, A., Bensalem, S., Bozga, M., Delahaye, B., Legay, A., Siffakis, E.: Verification of an AFDX Infrastructure using Simulations and Probabilities. In: Runtime Verification, RV'10. LNCS, vol. 6418. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
10. Batteux, M., Prosvirnova, T., Rauzy, A., Kloul, L.: The AltaRica 3.0 project for model-based safety assessment. In: 11th IEEE International Conference on Industrial Informatics, INDIN 2013, Bochum, Germany, July 29-31, 2013. pp. 741–746. IEEE (2013), https://doi.org/10.1109/INDIN.2013.6622976
11. Bensalem, S., de Silva, L., Griesmayer, A., Ingrand, F., Legay, A., Yan, R.: A Formal Approach for Incremental Construction with an Application to Autonomous Robotic Systems. In: Apel, S., Jackson, E.K. (eds.) Software Composition - 10th International Conference, SC 2011, Zurich, Switzerland, June 30 - July 1, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6708, pp. 116–132. Springer (2011), https://doi.org/10.1007/978-3-642-22045-6_8
12. Bittner, B., Bozzano, M., Cavada, R., Cimatti, A., Gario, M., Griggio, A., Mattarei, C., Micheli, A., Zampedri, G.: The xSAP Safety Analysis Platform. In: TACAS 2016. pp. 533–539 (2016)
13. Bittner, B., Bozzano, M., Cimatti, A., Ferluc, R.D., Gario, M., Guiotto, A., Yushtein, Y.: An Integrated Process for FDIR Design in Aerospace. In: IMBSA 2014. pp. 82–95 (2014)

14. Cherdantseva, Y., Burnap, P., Blyth, A., Eden, P., Jones, K., Soulsby, H., Stoddart, K.: A review of cyber security risk assessment methods for SCADA systems. Computers & security 56, 1–27 (2016)
15. David, A., Larsen, K., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Statistical Model Checking for Biological Systems. Int. J. Softw. Tools Technol. Transf. (STTT) 17(3), 351–367 (Jun 2015)
16. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: Uppaal SMC Tutorial. STTT 17(4), 397–415 (August 2015)
17. Desai, A., Qadeer, S., Seshia, S.A.: Programming Safe Robotics Systems: Challenges and Advances. In: International Symposium on Leveraging Applications of Formal Methods. pp. 103–119. Springer (2018)
18. Dragomir, I.: ESROCOS Planetary Exploration Demonstrator: the Watchdog component in TASTE and BIP. https://github.com/ESROCOS/control-mc_watchdog
19. Dragomir, I., Iosti, S., Bozga, M., Bensalem, S.: Designing Systems with Detection and Reconfiguration Capabilities: A Formal Approach. In: Steffen, B., Margaria, T. (eds.) Leveraging Applications of Formal Methods, Verification and Validation - 8th International Symposium, ISoLA 2018, Lymassol, Cyprus, November 5-9, 2018. Lecture Notes in Computer Science, Springer (november 2018)
20. Enrico, Z.: An introduction to the basics of reliability and risk analysis, vol. 13. World scientific (2007)
21. Foughali, M., Berthomieu, B., Dal Zilio, S., Hladik, P.E., Ingrand, F., Mallet, A.: Formal Verification of Complex Robotic Systems on Resource-Constrained Platforms. In: FormaliSE: 6th International Conference on Formal Methods in Software Engineering (2018)
22. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate Probabilistic Model Checking. In: International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI'04. pp. 73–84 (January 2004)
23. Risk management - guidelines. Standard, International Organization for Standardization, Geneva, CH (Feb 2018)
24. Jegourel, C., Legay, A., Sedwards, S.: Importance Splitting for Statistical Model Checking Rare Properties. In: CAV. vol. 13, pp. 576–591. Springer (2013)
25. Kahn, H., Marshall, A.W.: Methods of Reducing Sample Size in Monte Carlo Computations. Journal of the Operations Research Society of America 1(5), 263–278 (1953), http://www.jstor.org/stable/166789
26. Khalid, A., Kirisci, P., Khan, Z.H., Ghrairi, Z., Thoben, K.D., Pannek, J.: Security framework for industrial collaborative robotic cyber-physical systems. Computers in Industry 97, 132–145 (2018)
27. Kulkarni, V.G.: Introduction to Modeling and Analysis of Stochastic Systems. Springer New York (2011)
28. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: International Conference on Information Security and Cryptology. pp. 186–198. Springer (2005)
29. Mediouni, B.L., Nouri, A., Bozga, M., Dellabani, M., Legay, A., Bensalem, S.: $\mathcal{S}$BIP 2.0: Statistical Model Checking Stochastic Real-Time Systems. In: Lahiri, S.K., Wang, C. (eds.) Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11138, pp. 536–542. Springer (2018), https://doi.org/10.1007/978-3-030-01090-4_33
30. Mediouni, B.L., Nouri, A., Bozga, M., Legay, A., Bensalem, S.: Mitigating security risks through attack strategies exploration. In: International Symposium on Leveraging Applications of Formal Methods. pp. 392–413. Springer (2018)
31. Mitsch, S., Ghorbal, K., Vogelbacher, D., Platzer, A.: Formal verification of obstacle avoidance and navigation of ground robots. The International Journal of Robotics Research 36(12), 1312–1340 (2017)
32. Munoz, M., Montano, G., Wirkus, M., Hoeflinger, K., Silveira, D., Tsiogkas, N., Hugues, J., Bruyninckx, H., Dragomir, I., Muhammad, A.: ESROCOS: a Robotic Operating System for Space and Terrestrial Applications. In: Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA) 2017, Leiden, Netherlands, June 20-22, 2017 (june 2017)
33. Nouri, A., Mediouni, B.L., Bozga, M., Combaz, J., Bensalem, S., Legay, A.: Performance evaluation of stochastic real-time systems with the SBIP framework. International Journal of Critical Computer-Based Systems 8(3-4), 340–370 (2018), https://www.inderscienceonline.com/doi/abs/10.1504/IJCCBS.2018.096439
34. Ocon, J., Colemenero, F., Estremera, J., Buckley, K., Alonso, M., Heredia, E., Garcia, J., Coles, A., Coles, A., Martinez, M., Savas, E., Pommerening, F., Keller, T., Karachalios, S., Woods, M., Dragomir, I., Bensalem, S., Dissaux, P., Schach, A., Marc, R., Weclewski, P.: The ERGO framework and its use in planetary/orbital scenarios. In: International Astronautical Congress (IAC) 2018, Bremen, Germany, October 1-5, 2018 (october 2018)
35. Pariyani, A., Seider, W.D., Oktem, U.G., Soroush, M.: Dynamic risk analysis using alarm databases to improve process safety and product quality: Part ii–bayesian analysis. AIChE Journal 58(3), 826–841 (2012)
36. Pnueli, A.: The Temporal Logic of Programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57 (1977), https://doi.org/10.1109/SFCS.1977.32

37. Raman, B., Nouri, A., Gangadharan, D., Bozga, M., Ananda Basu, M.M., Legay, A., Bensalem, S., Chakraborty, S.: Stochastic Modeling and Performance Analysis of Multimedia SoCs. In: International conference on Systems, Architectures, Modeling and Simulation, SAMOS'13. pp. 145–154 (2013)
38. Roy, A., Kim, D.S., Trivedi, K.S.: Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. Security and Communication Networks 5(8), 929–943 (2012)
39. Tosun, T., Jing, G., Kress-Gazit, H., Yim, M.: Computer-aided compositional design and verification for modular robots. In: Robotics Research, pp. 237–252. Springer (2018)
40. Wander, A., Forstner, R.: Innovative Fault Detection, Isolation and Recovery Strategies On-board Spacecraft: State of the Art and Research Challenges. Deutscher Luft- und Raumfahrtkongress (2012)
41. Younes, H.L.S.: Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. thesis, Carnegie Mellon (2005)
42. Zio, E.: The future of risk assessment. Reliability Engineering & System Safety 177, 176 – 190 (2018), http://www.sciencedirect.com/science/article/pii/S0951832017306543

A Full $\mathcal{S}$BIP Model of the Case Study

A.1 Architecture

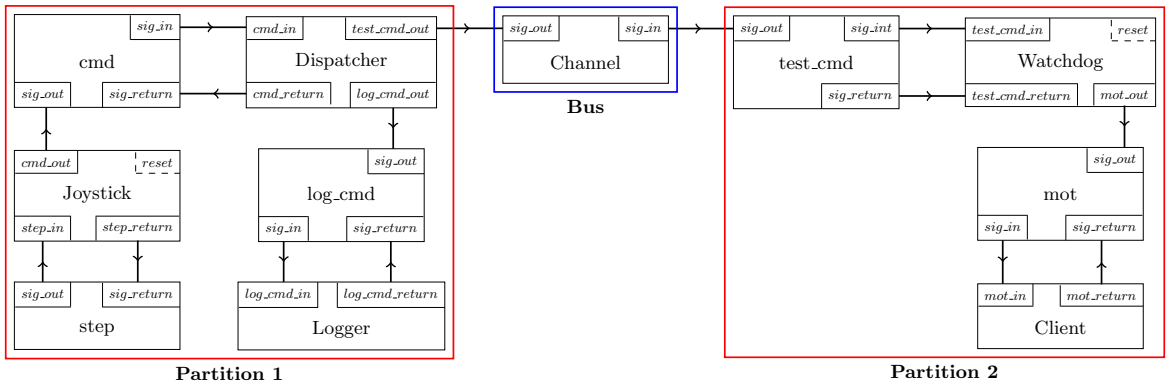


Fig. 15: Deployed architecture of the case study in BIP.

A.2 Behaviour

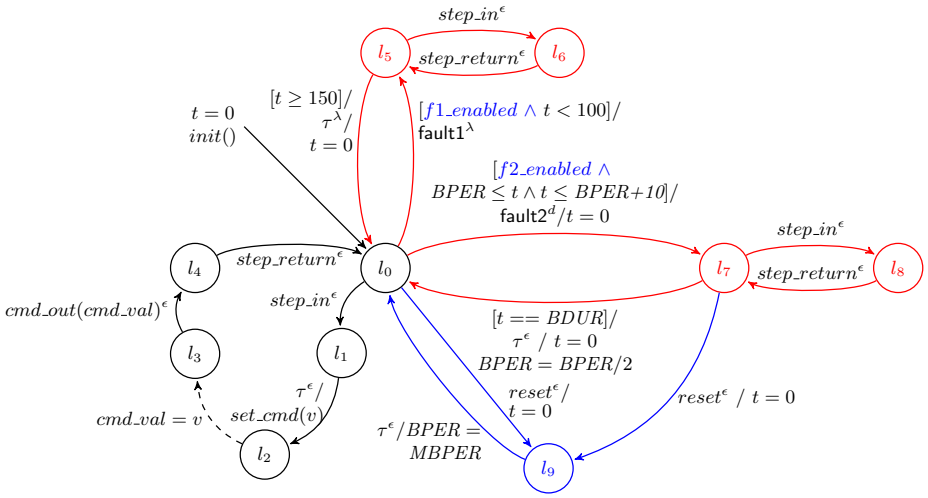


Fig. 16: Behaviour of Joystick

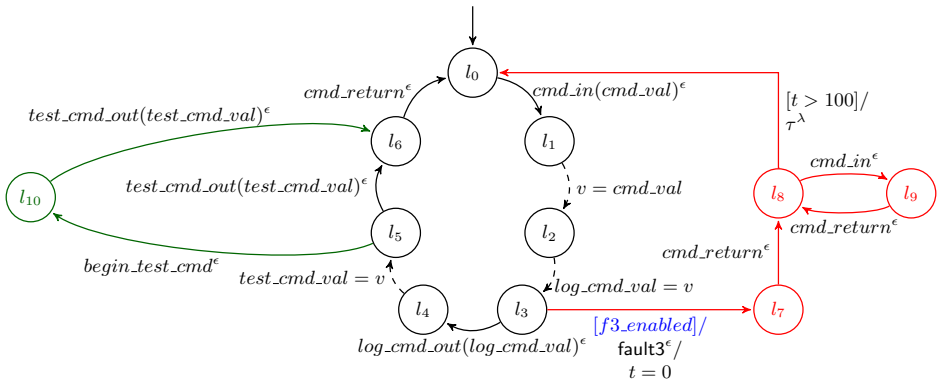


Fig. 17: Behaviour of Dispatcher

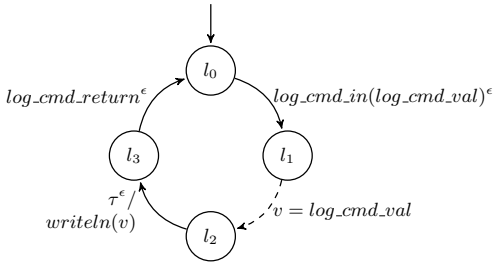


Fig. 18: Behaviour of Logger

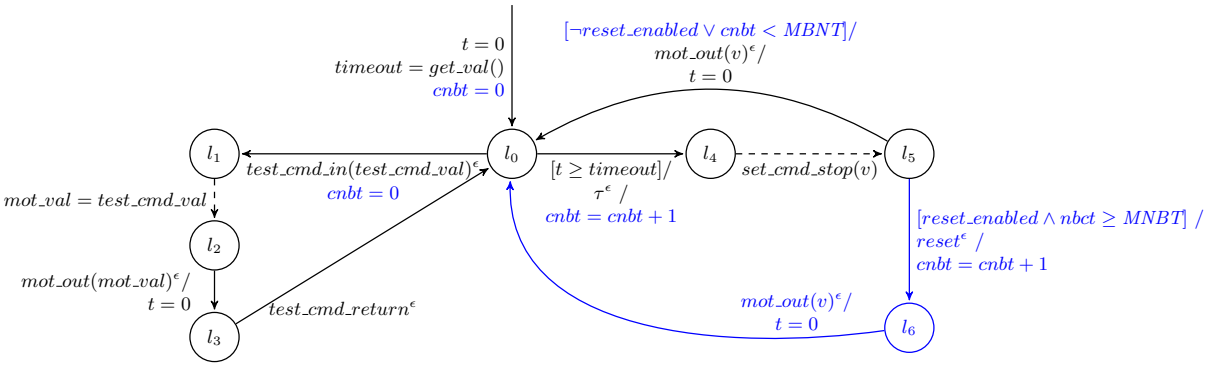


Fig. 19: Behaviour of Watchdog

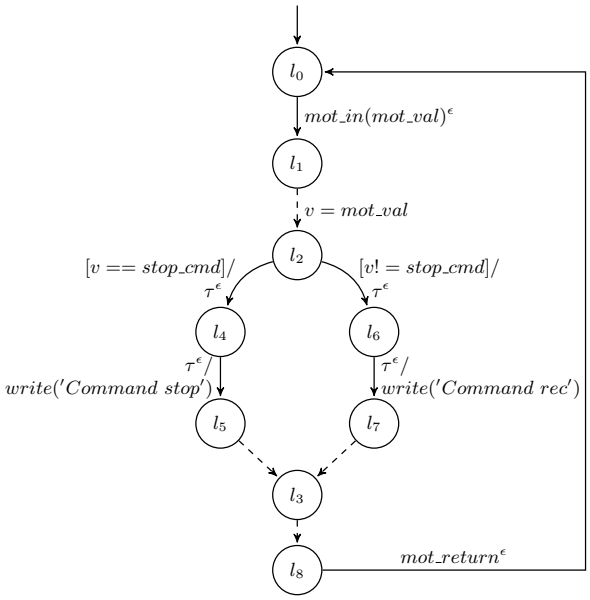


Fig. 20: Behaviour of Client

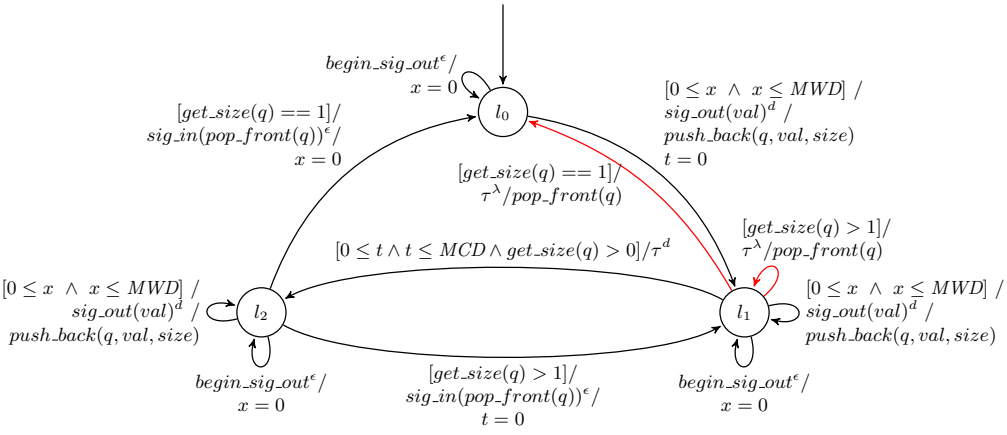


Fig. 21: Behaviour of Channel.