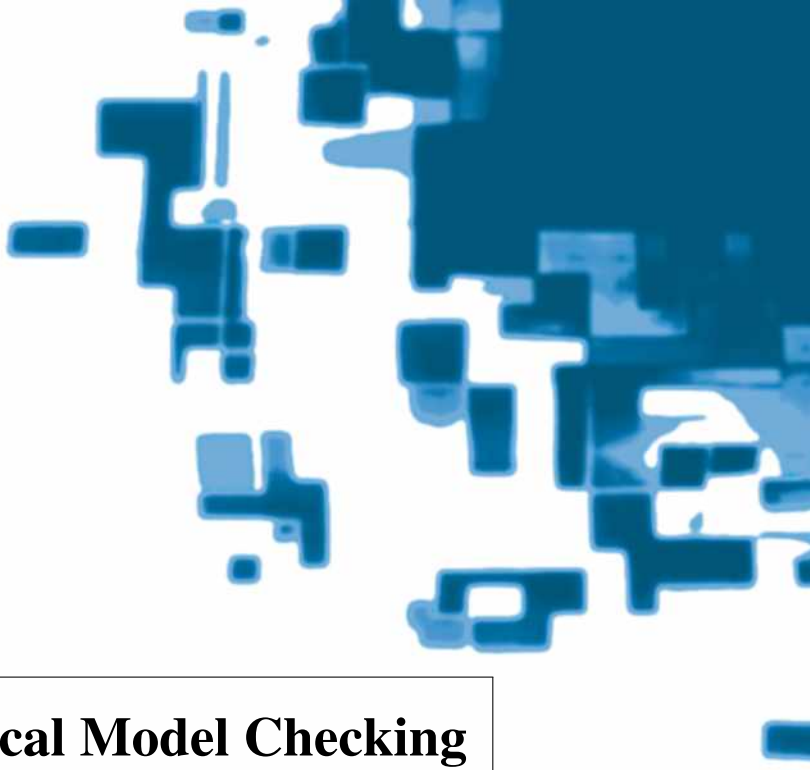


*S*BIP 2.0: Statistical Model Checking Stochastic Real-time Systems

Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Mahieddine Dellabani, Jacques Combaz, Axel Legay and Saddek Bensalem

Verimag Research Report n^o TR-2018-5

May 2, 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/

$\mathcal{S}$BIP 2.0: Statistical Model Checking Stochastic Real-time Systems

Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Mahieddine Dellabani, Jacques Combaz, Axel Legay and Saddek Bensalem

May 2, 2018

Abstract

This paper presents a major new release of $\mathcal{S}$BIP, an extensible statistical model checker for Metric (MTL) and Linear-Time Temporal Logic (LTL) properties on respectively Generalized Semi-Markov Processes (GSMP), Continuous-Time (CTMC) and Discrete-Time Markov Chain (DTMC) models. The newly added support for MTL, GSMPs and CTMCs allows to capture both real-time and stochastic aspects, enabling faithful specification and modeling of real-life systems. $\mathcal{S}$BIP was entirely redesigned as an IDE including project management, model edition, compilation, simulation, and statistical analysis. The tool has been used for various benchmarks and case studies including models of communication protocols, embedded and IoT systems.

Keywords: Stochastic Models, Statistical Analysis, Automated Verification

Reviewers: Marius Bozga

How to cite this report:

```
@techreport {TR-2018-5,
  title = {SBIP 2.0: Statistical Model Checking Stochastic Real-time Systems},
   author = {Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Mahieddine Dellabani,
Jacques Combaz, Axel Legay and Saddek Bensalem},
  institution = {{Verimag} Research Report},
  number = {TR-2018-5},
  year = {2018}
}
```

Contents

1 Introduction

Statistical Model Checking (SMC) is a powerful alternative to classical numerical probabilistic model checking techniques that fail to handle large state-space systems. SMC was successfully applied in the assessment of different real-life systems in various application domains [32, 16, 9, 7, 10]. Classical model checkers [26, 17, 4] now include SMC as part of their analysis engines, and have been recently joined by a variety of specialized SMC tools [35, 21, 8].

In this paper, we present the newest release of the $\mathcal{S}$BIP statistical model checker. In its previous version [28], the tool was restricted to the analysis of DTMC models with respect to bounded LTL properties. $\mathcal{S}$BIP now supports multiple modeling formalisms ranging from DTMCs to CTMCs and GSMPs. Moreover, it supports a parametric variant of MTL to allow expressing richer properties regarding time. Furthermore, the LTL support was extended towards nested temporal operators and boolean combination of properties.

From the user point of view, the tool has benefited from a major revision in its workflows. It now provides a single integrated environment where one can edit models, compile, simulate, and perform SMC analysis. For example, given a parametric MTL formula and an instantiation domain (for the property parameter), the tool performs automatically the necessary iterations for the analysis of different instances of the property. Moreover, to enhance the user experience, $\mathcal{S}$BIP is now organized around projects that enclose models, properties, traces, in a structured manner. It also includes support for visualization of obtained analysis results in standard formats, e.g., bar plots, charts.

We used $\mathcal{S}$BIP for modeling and analyzing several case studies including network protocols (Firewire, Bluetooth, PTP) and embedded systems (gear controller, pacemaker) with respect to various settings under different requirements, timed and untimed. We provide for all these examples, analysis results with a detailed evaluation of the tool performance.

Outline. Section 2 presents the $\mathcal{S}$BIP architecture and overviews its different functionalities and modules. We provide a brief presentation of the modeling and specification formalism in Section 3 before detailing the main modules in Section 4. In Section 5, we present different case studies addressed using the tool. A discussion of the related tools is presented in Section 6. Finally, Section 7 concludes and presents ongoing and future work directions. Further technical details and case studies are presented in Appendices.

2 $\mathcal{S}$BIP Design and Functionalities

The new version of $\mathcal{S}$BIP was completely re-designed as an IDE including all the activities from the modeling, to the simulation and the SMC analysis. It offers a set of functionalities organized in a clear and fluid workflow as illustrated in Fig. 1. All interactions with $\mathcal{S}$BIP go through a graphical user interface (GUI), which allows for setting the inputs, running analysis and getting the outputs.

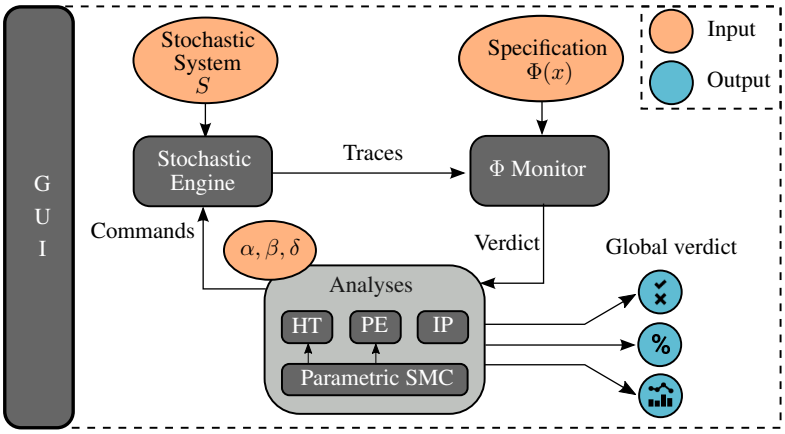


Figure 1: $\mathcal{S}$BIP architecture

2.1 Modular and extensible architecture.

The tool architecture was designed modularly for more flexibility and to enable extensibility. The tool relies on five main generic functional modules, namely, a *Stochastic Simulation Engine*, a *Monitoring* module, an *SMC Engine*, a *Rare-Event Engine*, a *Parametric Exploration* module plus additional data structures, i.e., to represent execution traces and logical formulas. The stochastic engine encapsulates an executable model simulator and is used to produce (random) execution traces on demand. The monitor is used to evaluate properties on traces. The SMC engine implements the main statistical model checking loop depending on the statistical method used, namely, hypothesis testing or probability estimation. Finally, the parametric exploration module coordinates the evaluation of a parametric property. All these modules are fully independent and interact through well-defined Java interfaces.

The tool has been instantiated for the BIP formalism as input model, where different stochastic simulation engines can be used (discrete/real-time). Regarding monitoring, the tool currently supports bounded LTL and parametric MTL. The functionality of these modules is detailed in Section 4.1 and Section 4.2.

2.2 Multiple integrated analysis workflows.

The tool takes as inputs a stochastic system model to be analyzed/simulated, a property of interest, and a set of parameters mainly required by the SMC algorithms. Three analysis workflows are provided by this new version. The first one is the classical SMC procedure consisting of either an *Hypothesis Testing* (HT) or a *Probability Estimation* (PE). It consists of triggering the stochastic simulation engine to produce a new execution trace which is monitored against the given property. This produces a local verdict, i.e., regarding that specific execution trace. Depending on the used SMC algorithm, several iterations are generally required to produce a global verdict. The second workflow consists of analyzing different instances of a parametric property by performing several iterations of the usual SMC workflow. The third workflow allows to verify rare properties on stochastic systems. These properties are subdivided into n intermediate properties of lower rarity and are represented as a scoring function. Details about the three workflows can be found in Sections 4.3, 4.4 and 4.5, respectively.

Depending on the used workflow, one can visualize the analysis results as a single probability, a yes/no answer, or a chart/bar plot. The tool also allows for visualizing the generated execution traces. Storing execution traces can be enabled/disabled by the user as it may be memory consuming. Further details on the features offered by the tool are provided in Section 4.6.

2.3 Technical info and availability.

$\mathcal{S}$BIP is fully developed in the Java programming language, and requires the Java Runtime Environment (JRE) 7. It uses ANTLR 4.7 [1] for LTL/MTL properties parsing, and the GNU Scientific Library (GSL) 2.3 library [2] for probability density functions manipulation. At this stage, $\mathcal{S}$BIP only runs on the GNU/Linux operating systems as it relies on BIP simulation engines. The tool is freely available for download at `http://www-verimag.imag.fr/Statistical-Model-Checking.html`.

3 Modeling and Specification Formalism

In this section, we provide a brief overview of the modeling and the specification formalisms supported natively by the $\mathcal{S}$BIP tool, namely the stochastic real-time BIP and the LTL/MTL logics.

3.1 Stochastic Real-time BIP

The *stochastic real-time* BIP formalism is an extension of the BIP formalism [11] developed at Verimag since more than ten years. BIP stands for Behavior-Interaction-Priority and provides concepts for modeling heterogeneous component-based systems using a layered approach. In BIP, systems are obtained by composition of atomic components (i.e., the behavior) with multiparty hierarchical interactions, and coordinated using dynamic priorities. BIP was mainly targeted for rigorous design of component-based systems, that is, not only formal modeling and analysis but also correct-by-construction implementation

and deployment. As such, BIP offers facilities to incorporate and execute external code within atomic components and/or multiparty interactions.

The stochastic real-time BIP formalism reconciles two distinct extensions of BIP that have been developed in the past. On the one hand, *real-time* BIP [3] extended BIP with real-time features (clocks, urgencies), has dense real-time semantics based on timed automata [5] with urgencies and is used for the modeling, the analysis and the implementation of real-time systems. On the other hand, *stochastic* BIP [28] extended BIP with stochastic features (probabilistic variables), has discrete-time stochastic semantics based on Markov chains and is mainly used for performing analysis using statistical model checking methods.

The stochastic real-time BIP [30] allows for defining components as timed automata extended with stochastic constraints. Such components are composed, under specific restrictions, using two categories of interactions, namely *timed* or *stochastic*. Timed interactions are associated only with (pure) timing constraints expressed as lower and upper bounds over clocks valuations, as in timed automata. These interactions are scheduled for execution with respect to an implicit uniform or exponential probability distribution as it is generally the case in several existing frameworks [17, 26]. Stochastic interactions are associated with one stochastic constraint (that is, a user-provided arbitrary density function), defining their occurrence time relative to a clock value. The underlying semantics of stochastic real-time BIP is defined as a Generalized Semi-Markov Process. It produces timed traces $\omega = (a_0, t_0)(a_1, t_1)\ldots$, where a_i are interactions and t_i are timestamps. A complete formal definition is available in [30].

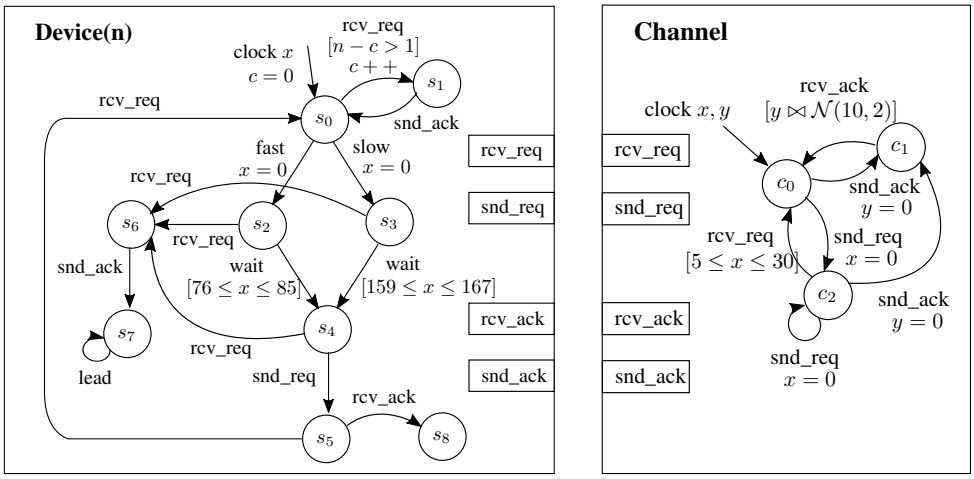


Figure 2: Stochastic real-time BIP: Components of the Firewire Protocol.

As an example, Fig. 2 provides a graphical illustration of stochastic real-time components. On the left, the Device component is essentially a timed automaton. On the right, the Channel component contains in addition a stochastic port *rcv_ack* defined by a normal density function, i.e., its scheduling time is sampled with respect to a normal function with mean 10 and standard deviation 2.

From the language point of view, stochastic interactions are defined using specific annotations @stochastic(dist="...", clk=..., param="...") to tag components ports. Such annotations specify a probability density function and its parameters through, respectively, the dist and param attributes, and associate a clock through the clk attribute. For example, the stochastic port *rcv_ack* of the Channel component in Fig. 2 is defined by dist="normal", clk=y, param="10,2" (see Example 1). Currently, the language supports a number of built-in density functions (normal, gamma, χ^2). Additional (empirical) functions, can be used through the same mechanism (dist="custom", and in param= a file characterizing the underlying cumulative distribution).

Example 1 *Below, we give an example of the stochastic real-time BIP modeling language representing the FireWire Channel component in Fig. 2.*

```
atom type Channel(int id)
  /* Data declaration */
  data int id_channel = id
  /* clocks declaration */
  clock x unit nanosecond
  clock y unit nanosecond
  /* ports declaration */
  export port ePort snd_ack( id_channel)
  export port Port snd_req()
  export port ePort rcv_req( id_channel)
  @stochastic(dist="normal",clk=y, param="10,2")
  export port Port rcv_ack()
  /* control locations */
  place c0, c1, c2
  /* transitions descriptions */
  initial to c0
  on snd_req from c0 to c2 do{x=0;}
  on snd_ack from c0 to c1 do{y=0;}
  on rcv_ack from c1 to c0
  on snd_req from c2 to c2 do{x=0;}
  on snd_ack from c2 to c1 do{y=0;}
  on rcv_req from c2 to c0 provided(x<=30 && x>=5)
end
```

3.2 Parametric MTL

Metric Temporal Logic (MTL) [24] is an expressive temporal logic that extends LTL by introducing an explicit representation of time. MTL temporal operators are similar to LTL with the difference of having a time interval $I \subseteq \mathbb{N}^+$ constraining the temporal operators. For a given Ψ, the set of (atomic) state formulas, the syntax of an MTL formula ϕ is inductively defined by the following grammar:

$$\phi ::= \boldsymbol{t} \mid \boldsymbol{f} \mid \psi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi \mid \phi_1 \, \mathcal{U}_I \, \phi_2 \mid \phi_1 \, \mathcal{R}_I \, \phi_2, \text{ where } \psi \in \Psi$$

The operator $\bigcirc \phi$ is the *next* operator, while $\phi_1 \, \mathcal{U}_I \, \phi_2$ is the *Until* operator, which stands for ϕ_1 holds until ϕ_2 does at any time in I. The *Release* operator $\mathcal{R}_I$ is the dual of $\mathcal{U}_I$. The *Eventually* and the *Globally* operators are expressed respectively as $\Diamond_I \phi \equiv \boldsymbol{t} \, \mathcal{U}_I \, \phi$, and $\Box_I \phi \equiv \boldsymbol{f} \, \mathcal{R}_I \phi$. Their meaning is respectively ϕ eventually holds at some time in I, and ϕ always holds at any time in I.

For the sake of usability, we allow expressing parametric MTL formula $\phi(x)$, where x is an integer parameter taking values in some bounded domain Π. The parameter can appear either in a state formula Ψ or as a bound of time intervals I and is statically assigned a value from its domain before starting analysis. For instance, $\phi(t) \equiv \Diamond_{[0,t]}[(node_3.status = leader)]$ states that $node_3$ eventually becomes the leader before t time units, where t is the parameter of the property.

4 $\mathcal{S}$BIP Modules

4.1 Stochastic Simulation Engine

The stochastic simulation engine implements the operational semantics of stochastic real-time BIP systems [30]. Given a model S, the engine produces system traces consistent with the timing and stochastic constraints in S. Traces are generated in two modes, namely, symbol-wise, or at-once for an *a priori* given length. The first is for online monitoring, since trace generation can be interrupted as soon as a verdict is obtained. The second is interesting in the context of SMC loop as it bounds the time for obtaining a local verdict.

The functioning of the stochastic simulation engine is depicted in Fig. 3. At every step, the engine computes the firing (time) interval for every interaction, based on current clock valuations and interaction guards (Evaluate). Next, an execution date is chosen for every future enabled interaction (Plan). For *timed interactions*, the date is chosen by sampling a value in the associated firing interval, using either a uniform or exponential law, depending if the firing interval is bounded or not. For *stochastic interactions*, the date

is chosen according to their associated probability density function and the clock value. Two cases are distinguished: when the current value of the clock is zero, the date is chosen by a direct sampling of the corresponding density. Otherwise, when the clock has a strict positive value, the execution date is planned using the truncated density function at that value [30]. Once all the future enabled interactions are planned, the scheduler implements a race policy and selects for execution the one having the earliest planned date. The simulation time is advanced to that date and the interaction is executed on the system (and logged).

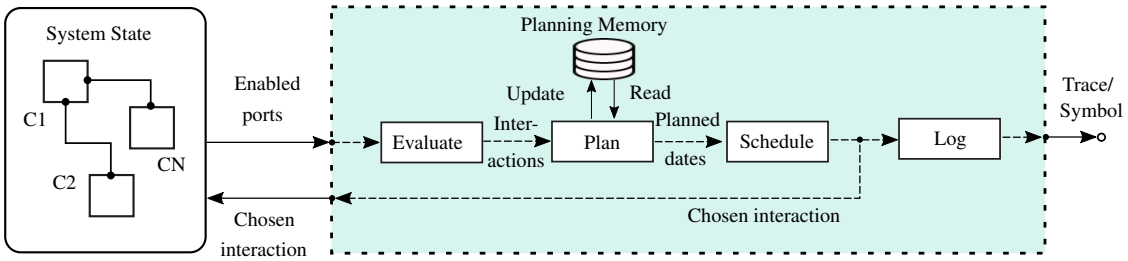


Figure 3: Functional view of the stochastic simulation engine

For efficiency reasons, planned execution dates are stored in order to avoid re-planning interactions that remain enabled when moving to the next system state. A new execution date is chosen only for newly enabled interactions and/or in conflict with the executed interaction. That is, when the associated clock (for stochastic interactions) has been reset, or the firing interval has changed due to execution of the previous interaction (see [30] for more details).

4.2 Monitoring Module

This module implements the generic infrastructure for online/offline monitoring of properties. At abstract level, the module takes as inputs a formula and either an entire trace or an online stream of trace symbols, and computes a verdict stating whether the trace satisfies the formula. Traces, formulas and symbols are designed as Java interfaces that can be extended with specific implementations.

In the current version of $\mathcal{S}$BIP , we integrate the monitoring of Bounded LTL and MTL formulas, untimed and timed BIP traces. Bounded LTL was already included in the first version of the tool. However, it was restricted to formulas without nested temporal operators. At contrary, the monitoring of MTL formulas represents a completely new development. The MTL monitor, illustrated in Fig. 4, implements an online monitoring algorithm based on the rewriting rules from [14]. Given an MTL formula ϕ and a timed trace ω, the monitor alternates rewriting and simplification phases. Rewriting consumes a timed symbol $\sigma_i = (a_i, t_i)$ of ω and partially evaluates the current formula ϕ into ϕ'. Partial evaluation includes the unfolding of temporal operators and evaluation of atomic state formulas to their truth value. Simplification applies reduction rules on the formula ϕ' based on Boolean logic (e.g., $(\boldsymbol{t} \wedge \phi') \equiv \phi'$) so as to conclude or to simplify it as much as possible before the next cycle.

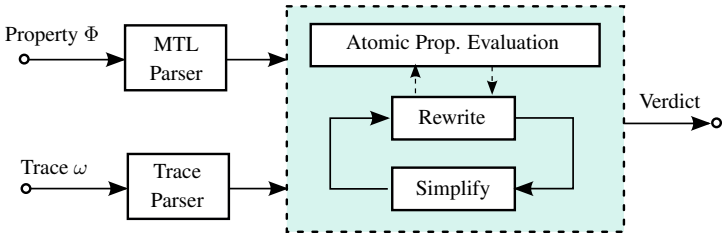


Figure 4: Functional view of the MTL Monitor

4.3 Statistical Model Checking Engine

The SMC engine implements several statistical testing algorithms for stochastic systems verification, namely, Single Sampling Plan (*SSP*), Simple Probability Ratio Test (*SPRT*) [33, 34], and Probability Estimation (*PESTIM*) [18]. We briefly recall the main procedures to decide whether a given stochastic system S satisfies a BLTL/MTL property ϕ. SMC refers to a series of simulation-based techniques to answer two types

of questions; **Qualitative:** is the probability for S to satisfy ϕ greater or equal to a certain threshold θ? and **Quantitative:** what is the probability for S to satisfy ϕ?

The main proposed approach to answer the qualitative question is based on *hypothesis testing* [34]. Let p be the probability of $S \models \phi$, to determine whether $p \geq \theta$, we can test $H : p \geq \theta$ against $K : p < \theta$. A test-based solution does not guarantee a correct result but it is possible to bound the probability of making an error. The *strength* (α, β) of a test is determined by two parameters, α and β, such that the probability of accepting K (resp., H) when H (resp., K) holds is less or equal to α (resp., β). Since it is impossible to ensure a low probability for both types of errors simultaneously, a solution is to use an *indifference region* $[p_1, p_0]$ (with θ in $[p_1, p_0]$) and to test $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$. Several hypothesis testing algorithms exist in the literature. [34] proposed a logarithmic-based algorithm that given p_0, p_1, α and β implements the *Sequential Ratio Testing Procedure* (*SPRT*) (see [33] for details). When one has to test $\theta \leq 1$ or $\theta \geq 0$, it is however better to use *Single Sampling Plan* (*SSP*) (see [12, 34] for details), an algorithm which the number of simulations is pre-computed in advance. In general, this number is higher than the one needed by *SPRT*, but is known to be optimal for the above-mentioned values. More details about hypothesis testing algorithms and a comparison between *SSP* and *SPRT* can be found in [12].

We also implement the estimation procedure (*PESTIM*) proposed in [18]. It enables to compute the probability p for S to satisfy ϕ. Given a *precision* δ, this procedure computes a value for p' such that $|p' - p| \leq \delta$ with *confidence* $1 - \alpha$. The procedure is based on the *Chernoff-Hoeffding bound* [20].

4.4 Parametric Exploration Module

Parametric exploration is an automatic way to perform statistical model checking on a family of properties that differ by the value of a constant. The family of properties is specified in a compact way as a parametric property $\phi(x)$, where x is an integer parameter ranging over a finite instantiation domain Π. Algorithm 1 illustrates the different phases of a parametric exploration workflow. The algorithm returns a set of SMC verdicts corresponding to the verification of the instances of $\phi(x)$ with respect to $x \in \Pi$. This workflow is very useful when exploring unknown system parameters such as, buffers sizes guaranteeing no overflow, or the amount of consumed energy. It automates the exploration for large parameters domains as opposed to tedious and time consuming manual procedures.

Data: system S, parametric property $\phi(x)$, instantiation domain Π
Result: A set of SMC verdicts V
Monitor m; Engine e; $V = \emptyset$;
foreach $v \in \Pi$ **do**
  smc.init();
  while *!smc.conclude()* **do**
    tr = *e.generate*(S);
    verdict = *m.check*($\phi(v)$, tr);
    smc.add(tr, *verdict*);
  end
  $V = V \cup$ *smc.getVerdict*() ;
end

Algorithm 1: Parametric exploration

4.5 Rare-Events Engine.

Importance Splitting (IP) [22] is a technique that allows decreasing the number of simulations required to estimate the probability of a rare event. The core of the technique is articulated around the ability to write the propery ϕ under study in an implicative form such that $\phi = \phi_n \Rightarrow \phi_{n-1} \Rightarrow ... \Rightarrow \phi_1$. Each ϕ_i represents a level l_i and the property is verified when all the n levels are satisfied. the probability of ϕ to be satisfied, denoted $\gamma = P(\rho \models \phi)$, is computed as the product of conditional probabilities to pass from level l_{i-1} to level l_i, i.e., $\gamma = P(\rho \models \phi) = \Pi_{i=1}^{n} P(\rho \models \phi_i \mid \rho \models \phi_{i-1})$. Note that the probability to be at level l_0 is relative to the probability to start a simulation at a given initial state in the model. Estimating the probability of a rare property is driven down to the estimation of n conditional probabilities that are less rare.

We represent the levels of a rare property as a scoring function $F : \Omega \longrightarrow \mathbb{N}$, that returns, for a given trace $\omega \in \Omega$, the highest level i of satisfied intermediate property ϕ_i. Hence, a trace ω satisties the property ϕ whenever $F(\omega) = n$.

4.6 Graphical User Interface

We implemented a user-friendly graphical interface (GUI) that centralizes all the interactions with the tool. The GUI is organized in three main regions: (1) a project explorer, (2) a toolbar and (3) a central panel as illustrated in Fig. 5.

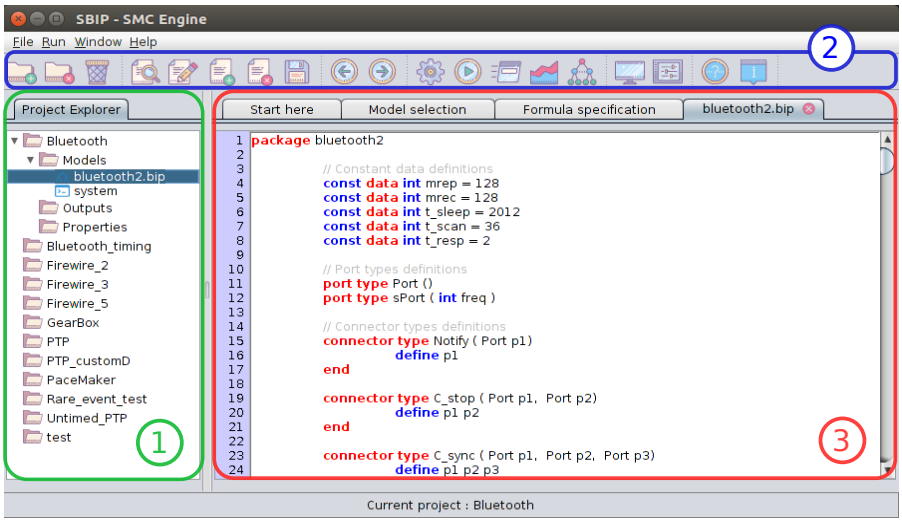


Figure 5: Screen-shot of $\mathcal{S}$BIP graphical user interface

The project explorer gives a centralized and organized view of the different items of a project during the modeling and the analysis. Such items usually include different files organized in a tree hierarchy. The *Models* folder contains (.bip) models, external (.cpp/hpp) source code, custom probability distributions, and executables. The *Properties* folder stores (.mtl) and (.ltl) properties. Finally, the *Outputs* folder contains execution traces.

The toolbar is organized in six functional groups: (i) project management, allowing to create/remove projects, (ii) file management for model files creation, deletion, edition and visualization, (iii) tab navigation, (iv) workflow management for model compilation, simulation and analysis, (v) configuration setup, and (vi) help buttons.

The central panel is the main region where the designer can load/visualize/edit inputs, configure parameters, run analyses, and visualize results. Each of these operations is provided through a specific view displayed in a separate tab:

- edition view: used to edit models and properties. This also allows for loading and saving various files. Specific capabilities, such as code auto-completion and keyword highlighting, are provided for BIP models.
- configuration view: used to select the simulation engine, the SMC algorithm and parameters, and the instantiation domain for parametric properties.
- analysis view: used to initiate and track the progress of analysis.
- results view: used to provide a summary of the performed analysis, the verdict and/or the set of verdicts on different traces, specific curves and/or plots, overall and partial running times, etc.

5 Case Studies and Benchmark

In this section, we present a set of case studies addressed using $\mathcal{S}$BIP, namely, the FireWire and the Bluetooth communication protocols, a vehicle gear controller, the Precision Time Protocol, and a pacemaker model, in addition to a concurrency model. Then, we also present an analysis of the tool performance. All these examples are available in the tool distribution.

5.1 FireWire – IEEE 1394

FireWire is a high-performance serial communication bus dedicated for hot plug-and-play multimedia devices. Devices can be organized in arbitrary topologies, where each pair of nodes is connected by two unidirectional channels. The internal representation of topologies is a tree where the root (*leader*) arbitrates the access to the bus. The designation of the leader is performed through a leader election protocol, namely, the *tree identification protocol*. Whenever the topology changes, i.e., a device joins/leaves, a reset occurs, and a new election is triggered.

The tree identification protocol is initiated by the leaf nodes of the topology. They send requests asking their neighbors to become their parents. A *parent request* sending mode is probabilistically determined to be $fast$ or $slow$. It determines the amount of time to wait before sending. Internal nodes of the topology keep on listening to parent requests until they receive exactly $n-1$ requests, n being the number of neighbors. Then, they send a parent request to their remaining neighbor. When receiving a parent request, a node either sends an acknowledgment, or detects a *contention* in the case it has also sent a parent request and it is still waiting for an acknowledgment. Intuitively, a contention means that two neighbors are mutually asking to be leader. This situation is resolved by forcing both nodes to send new requests after a random waiting time.

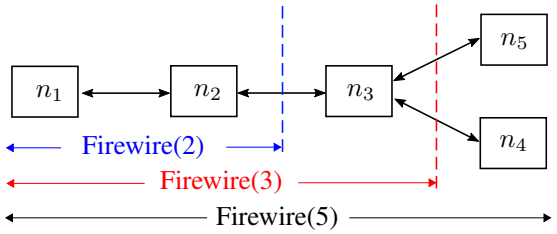


Figure 6: Considered Firewire topologies

We implemented a FireWire model inspired from the case-study in [15], where the considered topology is made of two devices. Our model is parametric, with m possible devices. We considered three particular topologies with $2, 3$ and 5 devices (Fig. 6). The models for a device and a channel are shown in Fig. 2. We studied the expected convergence time for the three topologies with ($\phi_1(t)$) and without ($\phi_2(t)$) contentions. We also investigated the topology impact on the probability of contentions (ϕ_3) and on the probability for each device (regarding its position) to be elected ($\phi_4(i)$). We provide the detailed MTL specifications of the properties verified on the FireWire model:

- the leader election procedure converges within t time units. It states that one of the nodes eventually becomes a leader and all the other nodes become slaves. The parametric exploration is used to find the expected time t^* when the process is guaranteed to converge (with probability 1).

$$\phi_1(t) \equiv \Diamond_{[0,t]} \bigvee_{i=1}^{m} [(node_i.s = 7) \bigwedge_{j=1, j\neq i}^{m} (node_j.s = 8)]$$

- the leader election procedure converges within t time units if no contention occurs. The property is basically an implication written as a disjunction of two parts. The first part of the disjunction is a conjunction between ϕ_1 and a second property stating that always no contention occurs during the election phase ($[0, t]$). The second handles the cases where a contention eventually occurs in $[0, t^*]$, where t^* is computed in ϕ_1. Note that the election and the contentions are detected at the level of the nodes.

$$\phi_2(t) \equiv (\Diamond_{[0,t]} \bigvee_{i=1}^{m} [(node_i.s = 7) \bigwedge_{j=1, j\neq i}^{m} (node_j.s = 8)]$$

$$\wedge \Box_{[0,t]} \bigwedge_{i=1}^{m} [\neg node_i.contention])$$
$$\vee (\Diamond_{[0,t^*]} \bigvee_{j=1}^{m} [node_i.contention])$$

- a contention eventually occurs during the election phase (t^* computed in ϕ_1):

$$\phi_3 \equiv \Diamond_{[0,t^*]} \bigvee_{i=1}^{m} (node_i.contention)$$

- the i^{th} device eventually becomes the leader (t^* computed in ϕ_1):

$$\phi_4(i) \equiv \Diamond_{[0,t^*]} (node_i.s = 7)$$

We used probability estimation with ($\alpha = 10^{-5}, \delta = 10^{-1}$) for all the analysis and relied on the parametric exploration to analyze properties $\phi_1(t)$ and $\phi_2(t)$.

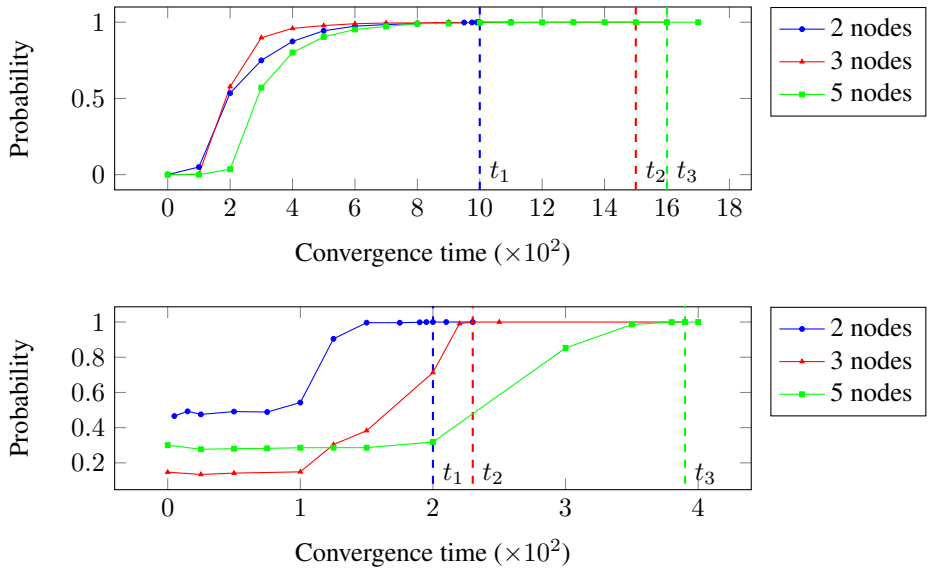


Figure 7: Probability of ϕ_1 (top) and ϕ_2 (bottom) for different FireWire topologies

We observed that the expected convergence time increases with larger topologies, as shown in Fig. 7 for $\phi_1(t)$ (top) and $\phi_2(t)$ (bottom). For $\phi_1(t)$, the expected time (in time units) was respectively 1000, 1500 and 1600 for the three considered topologies. When no contention occurs ($\phi_2(t)$), the expected time drops to 200, 230 and 390. The protocol spends more than 80% of the time resolving contentions. The analysis results for ϕ_3 and $\phi_4(i)$ are summarized in Table 1. We noticed that in a two-device topology, both nodes send parent requests almost simultaneously and thus have equal chances to become leader, but leads to $\sim$ 50% chance of contention. In larger topologies, leaf nodes initiate the election protocol, hence they have less chance to become leaders (nodes $n_{1,3}$ in FireWire(3) and $n_{1,4,5}$ in FireWire(5)). In contrast, inner nodes are more likely to become leader and this increases proportionally to the number of their neighbors. Moreover, we observed that the probability of contention in FireWire(3) is lower than the other topologies. Actually, contentions do not only depend on the number of nodes in the network but also on the way they are interconnected.

FireWire	ϕ_3	$\phi_4(1)$	$\phi_4(2)$	$\phi_4(3)$	$\phi_4(4)$	$\phi_4(5)$
(2)	0.493	0.507	0.493	-	-	-
(3)	0.137	0.042	0.92	0.038	-	-
(5)	0.289	0	0.4	0.6	0	0

Table 1: Results for properties ϕ_3 and ϕ_4

5.2 Bluetooth – Device Discovery

Bluetooth is a short-range wireless communication protocol for data exchange that promises low-energy consumption. A serious challenge in this protocol is interference. The Bluetooth standard relies on frequency hopping to tackle this issue. It allows devices to rapidly alternate among predefined frequency bands in a (pseudo-)random fashion. In order to perform data transfer, nodes in the network initially organize themselves into *piconets*, that is, small groups of one master and up to 7 slaves, where frequency hopping are synchronized. The *device discovery* phase lets one of the devices (called *inquiring*) become the master of the piconet by broadcasting messages to discover scanning devices, i.e., potential slaves. During the discovery phase, each node of the network can be in one of two modes (1) *active*, where it permanently looks to send or receive messages, and (2) *sniff*, where it alternates between sleeping and listening phases.

We built a model of the Bluetooth protocol (see Fig. 9), precisely the device discovery mechanism, based on the implementation in [6] that considers one receiver (slave) and one sender (master), where the receiver is set to the sniff mode. We improved [6] by considering a parametric model where the receiver can be in addition in the active mode. Fig. 8 represents the model of the Bluetooth components. The top figures show the receiver in active (left) and sniff (right) modes. The figures on the bottom show the frequency and the sender components from left to right respectively.

In active mode, the receiver sends a start signal to wake the sender component up and starts scanning the different frequencies through the frequency component. The receiver switches to a state where it is ready to receive $inquiries$ whenever it hears a transmission attempt on its frequency. Finally, the receiver commands the sender (and the frequency) component to stop by sending a stop signal and goes back to its initial state where other transmissions can be initiated by taking the dashed Stop transition.

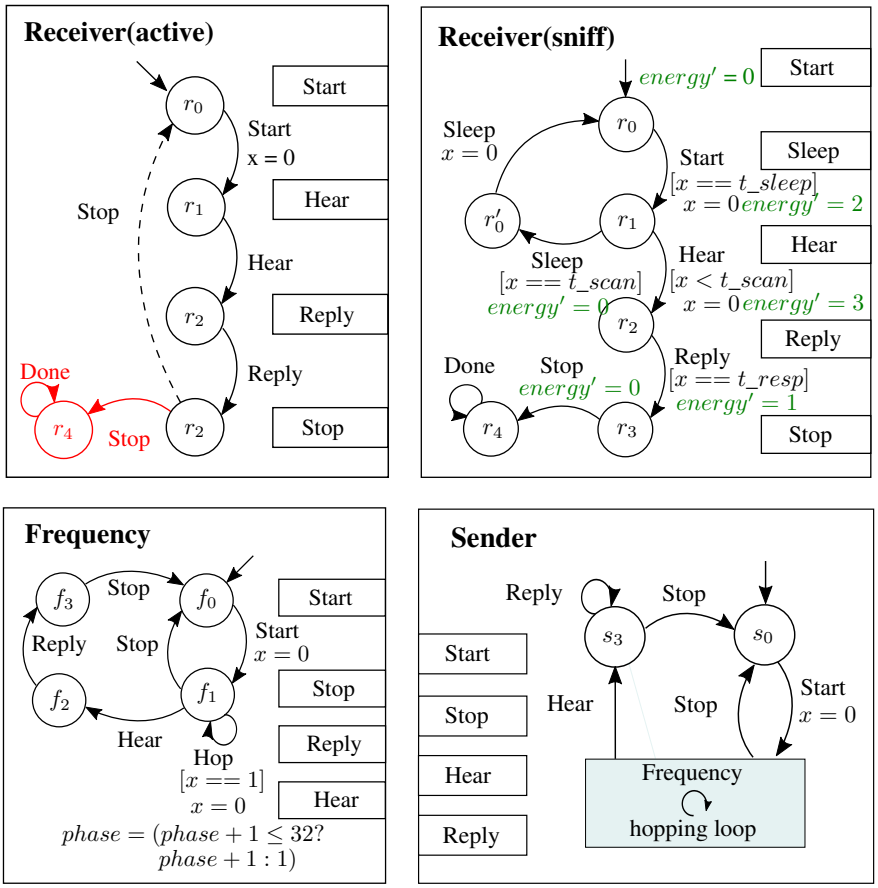


Figure 8: Components of the Bluetooth model

In the sniff mode, the receiver alternates between sleeping and scanning states. The sleeping phases of t_sleep (2012) time slots are followed by a scanning phase of t_scan (36) time slots, where the time slot is 0.3125 ms. During this scanning phase, the receiver can detect a transmission then replies to the $inquiry$ that requires t_resp (2) time slots to achieve. The receiver is also enriched with a cost clock that computes the amount of consumed energy (in energy units). In the sniff mode, no energy is consumed in sleep states (r_0, r'_0), whereas the receiver consumes 2 energy units per time slot in scan state r_1, and 3 units/time slot during the $reply$ (at state r_2).

We measure the impact of the different modes on the delays of discovery and on the receiver's energy consumption. In our model, the discovery process successfully terminates when the sender receives one reply ($sender.rec = 1$). We further model energy consumption of the receiver through a reward clock denoted $energy$. The first requirement is expressed in MTL as $\phi_5(t) \equiv \Diamond_{[0,t]}(sender.rec = 1)$ and states that the discovery must eventually occur within t time units. Our goal is to identify t^* that satisfies this requirement with probability 1. The second requirement is expressed as $\phi_6(e) \equiv \Box_{[0,t^*]}(receiver.energy \leq e)$. It states that the energy consumed by the receiver, during the discovery phase, is always under some threshold e. Again, the goal is to determine e^* that satisfies the requirement with probability 1. Both properties are expressed as parametric MTL and are assessed by using the parametric exploration.

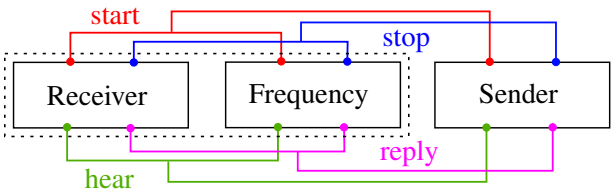


Figure 9: Bluetooth model with two devices

Fig. 10 summarizes the results obtained by applying the probability estimation algorithm with parameters ($\alpha = 10^{-3}, \delta = 10^{-1}$) for both modes. As expected, the active mode leads to a shorter discovery phase. On the left, we see that $t^* = 350$ ensures a convergence with probability 1. In the sniff mode (middle), the required time jumps to $t^* = 17000$. Regarding energy (right), in the active mode, the expected energy consumption of the receiver is $e^* = 700$ units, whereas it drops to $e^* = 600$ units when the receiver works in the sniff mode. That is, an energy saving of more than 14% compared to the active mode.

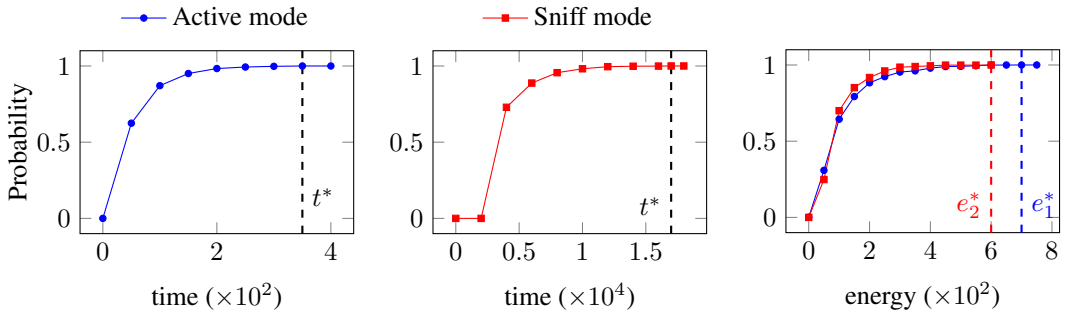


Figure 10: Probability of properties ϕ_5 (left and middle) and ϕ_6 (right)

5.3 A Vehicle Gear Controller

The Gear controller system is a real-time component embedded in modern cars. It is responsible for implementing the actual gear change requests issued by the driver (or by an algorithm) and transmitted through a communication network. The correctness and performance of the gear controller are important to guarantee a safe behavior of the vehicle. For instance, an excessive time for performing a gear change makes driving unpleasant but may

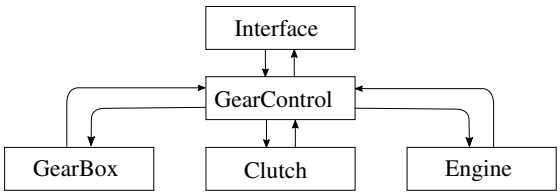


Figure 11: Gear controller model

also lead to serious safety problems.

We consider a stochastic real-time BIP model of this system based on the work in [27]. The model is composed of the gear controller component and its environment: a gear change request interface, a gear box, a clutch, and an engine (Fig. 11). Each of these components obeys to specific timing requirements. For instance, the Clutch can open or close within $100 - 150$ ms, the Gear box, which is electrically controlled, can set (resp. release) a gear in $100 - 300$ ms (resp. in $100 - 200$ ms). The engine operates in 3 modes with different constraints, i.e., normal, zero torque and synchronous speed. In the first mode, the engine gives the requested torque, in the second (resp third) it tries to find a zero torque (resp. speed) difference with the transmission (resp. the wheels). The maximum allowed time for searching a zero torque (resp. synchronous speed control) is 400 ms (resp. 500 ms). Missing any of these constraints raises errors in the model.

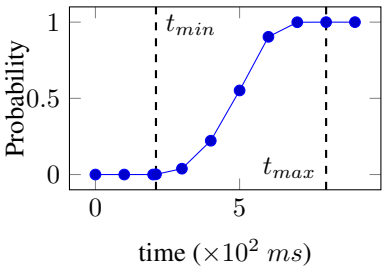


Figure 12: Probability of $\phi_7(t)$

In the original work, the authors used reachability analysis to prove several requirements concerning functional and performance aspects. We consider a subset of the original requirements (29 MTL properties). Here, we focus on those concerning the system performance. We provide results for one parametric property $\phi_7(t)$, which states that a complete gear change is always performed within t time units in the case of no errors. It is expressed in MTL as follows:

$$\phi_7(t) \equiv \Diamond_{[0,t]}[(\neg(gb.ErrStat = 0) \vee \neg(c.ErrStat = 0)$$

$$\vee\neg(e.UseCase = 0) \vee \ (gc.GearChanged) \vee (gc.Gear)) \wedge \neg(gc.SysTimer = 0)]$$

We used the probability estimation algorithm with parameters $(\alpha = 10^{-3}, \delta = 10^{-1})$. The obtained results using the parametric exploration are summarized in Fig. 12. We observe that the largest time value required to implement a gear change with probability 1 is 800 ms. In the same figure, we can also see that the shortest time with a non-zero probability for a gear change is 210 ms.

5.3.1 The complete set of considered requirements

The complete set of verified requirements is listed below (the required verification time is given between brackets). Note that some requirements are expressed as several MTL properties. For instance, requirement P_{13} induces 6 MTL properties. We also point out the fact that requirements $P_{9,10}$ were not considered since they address events that are very unlikely to occur. Note that Importance Splitting also fails to analyse them due to the impossibility to identify suitable levels (that are less rare).

- P_1. The gear can be changed. *[41s]* $\Diamond_{[0,1500]}$ $(gc.GearChanged)$
- P_2. The gear can be set to gear 5 and the reverse gear. *[16m 11s]*
  a. $\Diamond_{[0,1000000]}$ $(inf.gear = 5)$
  b. $\Diamond_{[0,1000000]}$ $(inf.gear = -1)$
- P_3. The switch gear can be performed in 1000 ms. *[42s]* $\Diamond_{[0,1500]}$ $(gc.GearChanged \wedge gc.SysTimer \leq 1000)$
- P_4. When the gearbox is in position N, the gear is zero. *[5m 16s]* $\Box_{[0,10000]}$ $((gb.Neutral \ \wedge inf.gear = 0) \ \vee \ \neg gb.Neutral \ \vee \ \neg inf.stableGear)$
- P_5. If the gearbox is idle then the gear is never N. *[10m 47s]*
  a. $\Box_{[0,10000]}$ $(\neg gb.Idle \ \vee \ \neg inf.N)$
  b. $\Box_{[0,10000]}$ $(\neg inf.N \ \vee \ gb.Neutral)$

- P_6. If there are no errors in gear and clutch and the engine is in normal mode, a gear switch is guaranteed in 900 ms, a switch gear can never be performed in less than 150 ms, and if the switch is not from/to gear N, a switch gear cannot be done in less than 400 ms. *[1m 26s]*

  a. a gear switch is guaranteed in 900 ms
  $\Diamond_{[0,900]}\ (\neg gb.ErrStat = 0\ \vee\ \neg c.ErrStat = 0\ \vee\ \neg e.UseCase = 0\ \vee\ gc.GearChanged)$

  b. A switch gear can never be performed in less than 150 ms
  $\Diamond_{[0,150]}\ (\neg gb.ErrStat = 0 \vee \neg c.ErrStat = 0 \vee \neg\ e.UseCase = 0 \vee \neg\ gc.GearChanged)$

  c. If the switch is not from/to gear N, a switch gear cannot be done in less than 400 ms
  $\Box_{[0,400]}\ (\neg gb.ErrStat = 0 \vee \neg c.ErrStat = 0 \vee \neg e.UseCase = 0 \vee \neg(inf.FromGear > 0\ \vee\ \neg inf.ToGear > 0\ \vee\ \neg gc.GearChanged)$

- P_7. If there are no errors in gear and clutch but engine in zero torque mode, a gear switch is guaranteed in 1055 ms, a switch gear can never be performed in less than 550 ms, and if the switch is not from/to gear N, a switch gear cannot be done in less than 700 ms. *[1m 30s]*

  a. a gear switch is guaranteed in 1055 ms
  $\Diamond_{[0,1055]}\ (\neg gb.ErrStat = 0\ \vee\ \neg c.ErrStat = 0\ \vee\ \neg e.UseCase = 1\ \vee gc.GearChanged)$

  b. switch gear can never be performed in less than 550 ms
  $\Diamond_{[0,550]}\ (\neg gb.ErrStat = 0 \vee \neg c.ErrStat = 0 \vee \neg e.UseCase = 1 \vee (\neg gc.GearChanged \wedge \neg gc.Gear))$

  c. If the switch is not from/to gear N, a switch gear cannot be done in less than 700 ms
  $\Box_{[0,700]}\ (\neg gb.ErrStat = 0\ \vee\ \neg c.ErrStat = 0\ \vee\ \neg e.UseCase = 1\ \vee\ \neg inf.FromGear > 0\ \vee\ \neg inf.ToGear > 0\ \vee\ \neg gc.GearChanged\ \vee\ \neg gc.Gear)$

- P_8. If there are no errors in gear and clutch but engine in synchronous speed mode, a gear switch is guaranteed in 1205 ms, a switch gear can never be performed in less than 450 ms, and if the switch is not from/to gear N, a switch gear cannot be done in less than 750 ms. *[1m 31s]*

  a. a gear switch is guaranteed in 1205 ms
  $\Diamond_{[0,1205]}\ (\neg gb.ErrStat = 0\ \vee\ \neg c.ErrStat = 0\ \vee\ \neg e.UseCase = 2\ \vee gc.GearChanged)$

  b. switch gear can never be performed in less than 450 ms
  $\Diamond_{[0,450]}\ (\neg\ e.UseCase = 2\ \vee\ (\neg gc.GearChanged\ \wedge\ \neg gc.Gear))$

  c. If the switch is not from/to gear N, a switch gear cannot be done in less than 750 ms
  $\Box_{[0,750]}\ (\neg gb.ErrStat = 0\ \vee\ \neg c.ErrStat = 0\ \vee\ \neg e.UseCase = 2\ \vee\ \neg inf.FromGear > 0\ \vee\ \neg inf.ToGear > 0\ \vee\ \neg gc.GearChanged\ \vee\ \neg gc.Gear)$

- P_{11}. The engine is guaranteed to find synchronous speed in the case where no error occurs in it. *[5m 52s]*

  $\Box_{[0,10000]}\ (\neg gb.ErrStat = 0\ \vee\ \neg c.ErrStat = 0\ \vee\ \neg e.isError)$

- P_{12}. If the gear is N, the engine is either in initial or going to initial (i.e. ToGear = 0 and engine in zero). *[5m 22s]*

  $\Box_{[0,10000]}\ (\neg inf.N\ \vee\ (inf.ToGear = 0\ \wedge\ e.Zero)\ \vee\ e.Initial)$

- P_{13}. Torque is always indicated in the engine when the gear controller has a gear set. *[31m 17s]*

  a. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = -1\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

  b. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = 1\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

  c. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = 2\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

  d. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = 3\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

e. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = 4\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

f. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = 5\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

- P_{14}. The controller is in predefined locations depending on the clutch state. *[10m 31s]*

  a. If clutch is open

  $\Box_{[0,10000]}\ (\neg c.Open \vee gc.ClutchOpen \vee gc.ClutchOpenTwo \vee gc.CheckGearSetTwo \vee gc.ReqSetGearTwo \vee gc.ClutchClose \vee gc.CheckClutchClosed \vee gc.CheckClutchClosedTwo \vee gc.CheckGearNeuTwo)$

  b. If clutch is closed

  $\Box_{[0,10000]}\ (\neg c.Closed \vee gc.ReqTorqueC \vee gc.GearChanged \vee gc.Gear \vee gc.Initiate \vee gc.CheckTorque\ \vee\ gc.ReqNeuGear\ \vee\ gc.CheckGearNeu\ \vee\ gc.ReqSyncSpeed\ \vee\ gc.CheckSyncSpeed\ \vee\ gc.ReqSetGear\ \vee\ gc.CheckGearSet)$

- P_{15}. The controller is in predefined locations depending on the gearbox status. *[10m 32s]*

  a. If gear is idle

  $\Box_{[0,10000]}\ (\neg c.Open \vee gc.ClutchOpen \vee gc.ClutchOpenTwo \vee gc.CheckGearSetTwo \vee gc.ReqSetGearTwo \vee gc.ClutchClose \vee gc.CheckClutchClosed \vee gc.CheckClutchClosedTwo \vee gc.CheckGearNeuTwo)$

  b. If gear is neutral

  $\Box_{[0,10000]}\ (\neg gb.Neutral \vee gc.ReqSetGear \vee gc.CheckClutchClosedTwo \vee gc.ReqTorqueC \vee gc.GearChanged \vee gc.Gear \vee gc.Initiate \vee gc.ReqSyncSpeed \vee gc.CheckSyncSpeed \vee gc.ReqSetGear\ \vee\ gc.CheckClutch\ \vee\ gc.ClutchOpen\ \vee\ gc.ReqSetGearTwo)$

- P_{16}. If engine regulates on torque, then the clutch must be closed. *[6m 08s]*

  $\Box_{[0,10000]}\ (\neg e.Torque\ \vee\ c.Closed)$

5.4 Precision Time Protocol – IEEE 1588

In this study, the Precision Time Protocol (PTP) is deployed as part of a distributed heterogeneous communication system in an aircraft [9] to synchronize the clocks of the different devices of the system. The reference clock is given by a specific device in the network called Master. This synchronization is essential to guarantee a correct behavior of the whole system.

We consider an abstract stochastic model of the PTP protocol shown in Fig. 13. It is composed of a master and a slave in addition to two communication channels. The considered model is parametric as it represents the communication of the master with different slaves of the actual system. This is expressed through different stochastic communication delays of the channels. Concretely, different probability density functions, depending on the position of the slave in the network. The stochastic behavior a Channel and the deterministic behavior of the Slave components. Additional details on the models can be found in [9].

An important property to verify on the system is that the drift between the clock of the master denoted tm and the clock of any slave denoted ts is always bounded by a threshold Δ. This property is expressed as $\phi_8(\Delta) \equiv \Box_{[0,T]}\ (abs(master.tm - slave.ts) \leq \Delta)$, where T is the period of the PTP protocol and $abs()$ is the absolute value function. Fig. 14b shows that the smallest bound Δ guaranteeing the property ϕ_8 is $\Delta^* = 70$. We used the probability estimation algorithm with $\alpha = 10^{-5}, \delta = 10^{-1}$ combined with the parametric exploration for the analysis of this property.

5.5 Pacemaker Model

A pacemaker is a device implanted on a human heart to cope with malfunctions due to aging or diseases. Its function is to guarantee the timed relationships between atrial and ventricular contractions. This device (see Fig. 15) acts as a monitor for these atrial and ventricular events and generates electrical pulses to compensate missing/late events and hence, prevents the heart's malfunctions.

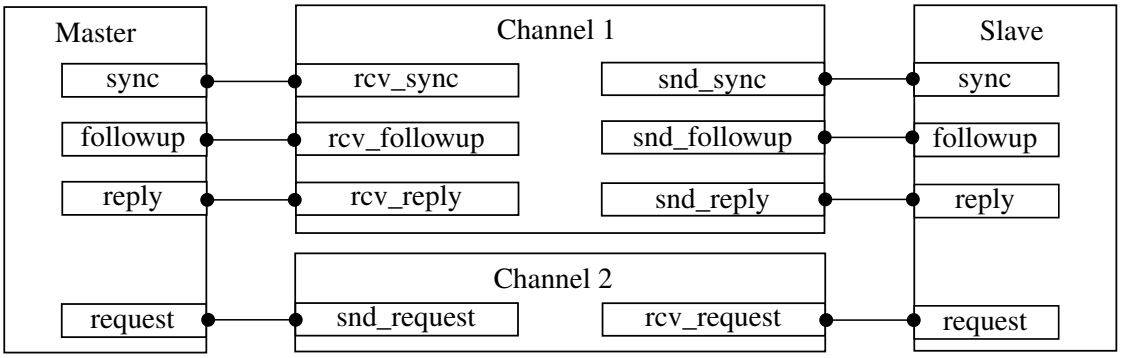
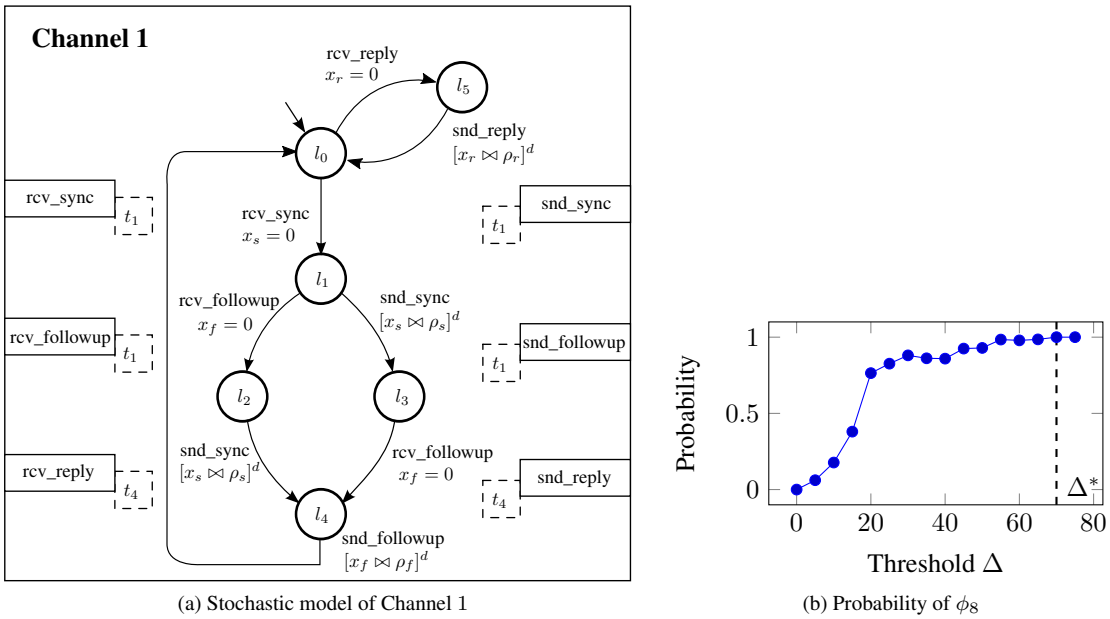


Figure 13: The abstract PTP model.



(a) Stochastic model of Channel 1

(b) Probability of ϕ_8

Figure 14: Stochastic model and analysis results

Our model is a BIP translation of the case study in [23]. In this model, the heart is represented by a component that periodically sends atrial and ventricular contraction events, respectively denoted AS and VS. These events are handled by the pacemaker that may deliver atrial pacing (AP) or ventricular pacing (VP) to regulate the heart timed behavior. The pacemaker is a compound component composed of four components: (i) *Lower Rate Interval (LRI)* ensures that the heart rate is above a minimum value by monitoring the elapsed time between ventricular events (VS, VP), generating AP if a time limit of TLRI-TAVI is reached. (ii) *Atrio-Ventricular Interval and Upper Rate Interval (AVIURI)* guarantees the delay between an atrial event and a ventricular one by delivering a ventricular pacing in the case where no ventricular contraction is detected within TAVI. This module also tracks delays between ventricular events to avoid pacing the ventricle too fast. (iii) *Post Ventricular Atrial Refractory Period (PVARP) and Post Ventricular Atrial Blanking (PVAB)* filters noise by ignoring atrial events occuring in TPVARP. (iv) *Ventricular Refractory Period (VRP)* ensures a minimum delay TVRP between ventricular events.

Parameter	Value
TLRI	1000
TAVI/TVPR	150
TURI/A_{min}	400
TPVAB	50
TPVARP/V_{min}	100
V_{max}	200
A_{max}	1100

Table 2: Parameters for the pacemaker and the heart models

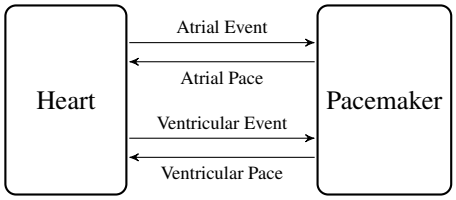


Figure 15: Heart and Pacemaker interactions

On the one hand, the pacemaker has to monitor the heart rate and verify that the interval between ventricular events is bounded by TLRI (property ϕ_9). On the other hand, it must not deliver VP too fast. This amounts to verify that the interval between a ventricular event and VP is above TURI (property ϕ_{10}).

We checked both properties on $\mathcal{S}$BIP using the probability estimation algorithm with parameters ($\alpha = 10^{-5}, \delta = 10^{-1}$). The analysis required 4883 execution traces, each one representing a simulation time of approximatively 8 minutes. Both properties have been proven true ($P(\phi_9) = P(\phi_{10}) = 1$) in less than $1h30$ per property.

5.6 Concurrency model

Concurrency is a key concept in systems in general, and programs in particular. Concurrent systems are the ones that can execute independantly which can lead to improve the overall execution-time of the systems tasks. One of the most common way to synchronize/communicate this kind of systems is through shared resources. However, one wants to study the fairness in the access to these shared resources.

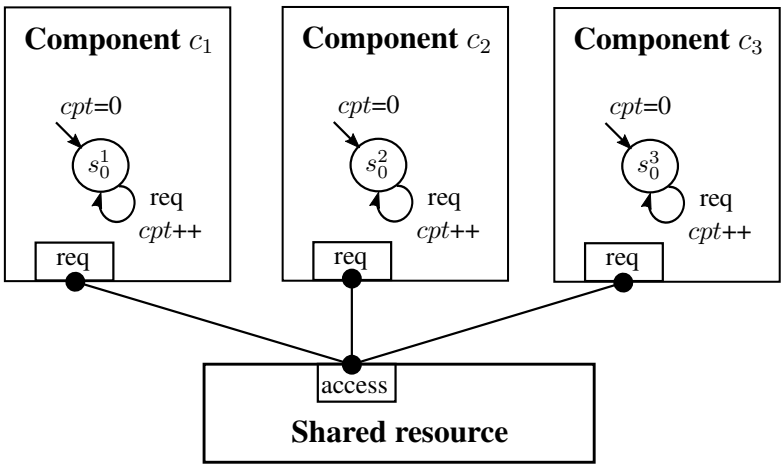


Figure 16: A concurrency model with three components sharing a single resource

In this case study, we consider three concurrent components that share a common resource, as depicted in Figure 16. Each component c_i memorizes the number of times it accessed the critical resource in an integer variable cpt. The considered LTL property $\phi_{11} \equiv F\{30\}(\bigwedge_{i=1}^{3}(c_i.cpt > 9))$ states that, after 30 system steps, the components access more than 9 times to the shared resource. For ϕ_{11} to be evaluated to true, each component must have exactly 10 accesses for an overall number of 30 accesses, corresponding to the 30 system steps, which makes the property rare. The decomposition into $n = 10$ levels can be done in a straightforward manner, that is, $l_k \equiv \bigwedge_{i=1}^{3}(c_i.cpt > k - 1), k \in [1, 10]$. This comes from the fact that we want the component to access the resource exactly the same number of times. The stop condition is hence set to 3 steps (one for each component).

We first tried to estimate the probability of property ϕ_{11} using PE with the parameters ($\delta = 0.03, \alpha = 0.001$). This lead us to simulate 33782 traces in which the rare event has never been met ($P(\phi_{11}) = 0$), in an overall execution time of 3m 37s. However, by using IP with $M = 1000$ traces we were able to

Probability	$P(l_1\|l_0)$	$P(l_2\|l_1)$	$P(l_3\|l_2)$	$P(l_4\|l_3)$	$P(l_5\|l_4)$
Estimate	0.215	0.234	0.217	0.222	0.227

Probability	$P(l_6\|l_5)$	$P(l_7\|l_6)$	$P(l_8\|l_7)$	$P(l_9\|l_8)$	$P(l_{10}\|l_9)$	$P(\phi_{11})$
Estimate	0.218	0.194	0.209	0.199	0.243	2.35×10^{-7}

Table 3: Results of IP on the concurrency model

estimate the probability of each level to occur and hence $P(\phi_{11})$, in less than 13s. Table 3, summarizes the results of IP on the concurrency model. We can see that the probability that concurrent components access exactly the same number of times the shared resource is very low $P(\phi_{11}) = 2.35 \times 10^{-7}$ but not null. It is interesting to see that the conditional probabilities are very close which indicates that the actual decomposition in levels is suitable, and hence reduces the relative variance of the final estimate.

5.7 Performance Analysis

We now provide performance measures of $\mathcal{S}$BIP, mainly regarding time (Table 4). The first three columns show respectively the considered case study, the number of components in the associated model, and the properties under test. The two remaining columns illustrate the number of SMC loops in case of parametric exploration and the average SMC time. We observe that depending on the model size and the property complexity, the SMC time can reach a dozen of minutes.

To get more insights on the reasons of the observed times, we investigated the individual tasks within an SMC loop, i.e., property parsing, trace simulation, trace parsing, and monitoring. We considered the processing time of a single execution trace (with lengths ranging between 10^5 and $2.\,10^5$) of the PTP model and one MTL property (see Appendix 5.4). Fig. 17 summarizes the obtained results, which show that the overall processing time grows linearly with the trace size. A noticeable observation is that the simulation time takes almost 90% of the whole analysis. This is mainly due to current prototype implementation of the stochastic real-time engine. Moreover, in this experiment, we systematically logged model variables, which considerably increased the simulation time. Related to this matter, we observe that the trace parsing (including instantiation) is also substantial. The reason for that is mainly strings manipulation. It grows proportionally with the size of the trace. Finally, we notice that the MTL parsing and monitoring require relatively short time and are almost constant in this case, since we considered the same property.

Case study	$\#C$	ϕ	#smc loops	avg smc time
Firewire(2)	4	ϕ_1	11	1m 21s
		ϕ_2	9	1m 59s
		ϕ_3	-	2m 28s
		ϕ_4	2	3m 27s
Firewire(3)	7	ϕ_1	17	1m 53s
		ϕ_2	11	3m 34s
		ϕ_3	-	3m 38s
		ϕ_4	3	4m 43s
Firewire(5)	13	ϕ_1	18	3m 54s
		ϕ_2	17	12m 36s
		ϕ_3	-	7m 23s
		ϕ_4	5	10m 16s
Bluetooth v1	3	ϕ_5	9	2m 27s
		ϕ_6	16	3m 11s
Bluetooth v2	3	ϕ_5	11	3m 0s
		ϕ_6	14	13m 05s
Gear Control	5	ϕ_7	11	54s
PTP	4	ϕ_8	15	8m 42s
Pacemaker	5	ϕ_9	-	1h 28m
		ϕ_{10}	-	1h 30m

Table 4: Summary of performance

6 Related Work

Several tools implement SMC analysis [21, 31, 26, 17, 13]. Some provide, in addition, distributed versions of the statistical tests, like [4, 21, 35, 19]. The main difference between these tools is the modeling and

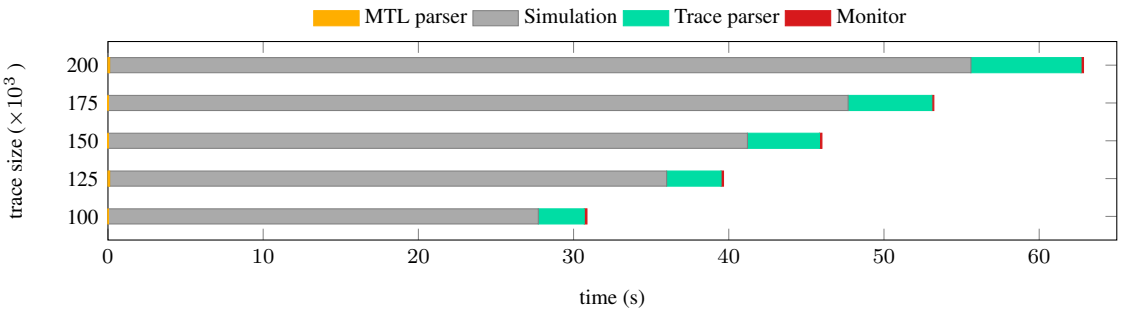


Figure 17: Detailed processing times for different trace sizes

specification formalisms, and sometimes the implemented statistical test, such as considering rare events.

Well-known tools such as UPPAAL-SMC [17] and PRISM [26] support various formalisms. The former considers *Networks of Priced Timed Automata* (NPTAs), which are high-level representations of CTMCs/DTMCs for system modeling, and weighted MTL for properties specification. Prism considers in addition *Markov Decision Processes* (MDPs) and Probabilistic Timed Automata (PTAs) for modeling, and *Probabilistic Computation Tree Logic* (PCTL), *Continuous Stochastic Logic* (CSL), and LTL, for properties specification. Other tools like Vesta [31] also support algebraic specification languages like PMaude [25].

Our tool was extended to support GSMPs and CTMCs using the stochastic real-time BIP formalism [30]. As opposed to the aforementioned tools, it allows for using arbitrary probability density functions. Except Ymer [35], the first to implement sequential hypothesis testing (unfortunately no more maintained), we are not aware of other tools offering such a general model. Furthermore, our tool provides high-level modeling mechanisms such as multiparty interactions (i.e., $n-$ary synchronizations) and events urgencies (e.g., lazy). It also allows for integrating external (legacy) code, which is not always the case in other tools. Finally, it is worth mentioning that $\mathcal{S}$BIP can also be used to generate (distributed) implementations ready for deployment on real platforms [29].

Similarly to our tool, PlasmaLab [21] is a modular and extensible statistical model checker that may be extended with external simulators and checkers. The default configuration supports DTMCs specified in a variant of the Prism language and requirements expressed in bounded LTL.

7 Conclusion

We presented the release 2.0 of the $\mathcal{S}$BIP tool, which offers new capabilities regarding the modeling and specification formalism, the analysis workflows, and the whole user experience. We added support for GSMPs and CTMCs and for MTL in addition to DTMCs and BLTL. We revisited the workflow of the tool to offer an integrated environment for design, analysis and visualization, in addition to new features such as the parametric exploration. We considered several case studies to show the tool's analysis capabilities and to assess its performance.

We identified through these investigations a bottleneck due to the current version of the simulation engine. We are currently working on an optimized version which can significantly improve the whole performance of the tool. Other interesting future directions are to extend the capabilities of the tool by integrating a distributed implementation of SMC.

References

[1] ANTLR Web page. http://www.antlr.org/. Accessed: 2017-10-11. 2.3

[2] GSL Web page. https://www.gnu.org/software/gsl/. Accessed: 2017-10-11. 2.3

[3] T. Abdellatif, J. Combaz, and J. Sifakis. Rigorous implementation of real-time systems - from theory to application. *Mathematical Structures in Computer Science*, 23(4):882–914, 2013. 3.1

[4] M. AlTurki and J. Meseguer. PVeStA: A parallel statistical model checking and quantitative analysis tool. In *Proceedings of the 4th International Conference on Algebra and Coalgebra in Computer Science, CALCO'11*, August 2011. 1, 6

[5] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, Apr. 1994. 3.1

[6] S. Arry and A. Kaur. Article: Formal verification of device discovery mechanism using uppaal. *International Journal of Computer Applications*, 58(19):32–37, November 2012. 5.2

[7] R. Balaji, A. Nouri, D. Gangadharan, M. Bozga, M. M. Ananda Basu, A. Legay, S. Bensalem, and S. Chakraborty. Stochastic modeling and performance analysis of multimedia socs. In *International conference on Systems, Architectures, Modeling and Simulation, SAMOS'13*, pages 145–154, 2013. 1

[8] P. Ballarini, B. Barbot, M. Duflot, S. Haddad, and N. Pekergin. HASL: A new approach for performance evaluation and model checking from concepts to experimentation. *Performance Evaluation*, 90:53–77, Aug. 2015. 1

[9] A. Basu, S. Bensalem, M. Bozga, B. Caillaud, B. Delahaye, and A. Legay. Statistical abstraction and model-checking of large heterogeneous systems. In *Forum for fundamental research on theory, FORTE'10*, volume 6117 of *LNCS*, pages 32–46. Springer, 2010. 1, 5.4

[10] A. Basu, S. Bensalem, M. Bozga, B. Delahaye, A. Legay, and E. Siffakis. Verification of an AFDX infrastructure using simulations and probabilities. In *Runtime Verification, RV'10*, volume 6418 of *LNCS*. Springer, 2010. 1

[11] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, SEFM'06, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society. 3.1

[12] S. Bensalem, B. Delahaye, and A. Legay. Statistical model checking: Present and future. In *RV*, volume 6418 of *LNCS*. Springer, 2010. 4.3

[13] J. Bogdoll, L. M. F. Fioriti, A. Hartmanns, and H. Hermanns. Partial order methods for statistical model checking and simulation. In *Forum for fundamental research on theory, FMOODS/FORTE'11*, pages 59–74, June 2011. 6

[14] P. E. Bulychev, A. David, K. G. Larsen, A. Legay, G. Li, and D. B. Poulsen. Rewrite-based statistical model checking of wmtl. *RV*, 7687:260–275, 2012. 4.2

[15] A. David, K. Larsen, A. Legay, M. Mikučionis, and Z. Wang. Time for statistical model checking of real-time systems. In *Computer Aided Verification*, pages 349–355. Springer, 2011. 5.1

[16] A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, and S. Sedwards. Statistical model checking for biological systems. *Int. J. Softw. Tools Technol. Transf.*, 17(3):351–367, June 2015. 1

[17] A. David, K. G. Larsen, A. Legay, M. Mikuăionis, and D. B. Poulsen. Uppaal smc tutorial. *Int. J. Softw. Tools Technol. Transf. (STTT)*, 17(4):397–415, August 2015. 1, 3.1, 6

[18] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI'04*, pages 73–84, January 2004. 4.3

[19] T. Herault, R. Lassaigne, and S. Peyronnet. APMC 3.0: Approximate verification of discrete and continuous time markov chains. In *Proceedings of the 3rd international conference on the Quantitative Evaluation of Systems*, QEST '06, pages 129–130, Washington, DC, USA, 2006. IEEE Computer Society. 6

[20] W. Hoeffding. Probability inequalities. *Journal of the American Statistical Association*, 58:13–30, 1963. 4.3

[21] C. Jegourel, A. Legay, and S. Sedwards. A platform for high performance statistical model checking — plasma. In *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'12, pages 498–503, Berlin, Heidelberg, 2012. Springer-Verlag. 1, 6

[22] C. Jegourel, A. Legay, and S. Sedwards. Importance splitting for statistical model checking rare properties. In *CAV*, volume 13, pages 576–591. Springer, 2013. 4.5

[23] Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and verification of a dual chamber implantable pacemaker. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 188–203, 2012. 5.5

[24] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, Nov 1990. 3.2

[25] N. Kumar, K. Sen, J. Meseguer, and G. Agha. A rewriting based model for probabilistic distributed object systems. In E. Najm, U. Nestmann, and P. Stevens, editors, *FMOODS*, pages 32–46, 2003. 6

[26] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: verification of probabilistic real-time systems. In *Proceedings of the 23rd international conference on Computer aided verification*, CAV'11, pages 585–591, Berlin, Heidelberg, 2011. Springer-Verlag. 1, 3.1, 6

[27] M. Lindahl, P. Pettersson, and W. Yi. Formal design and analysis of a gear controller. *International Journal on Software Tools for Technology Transfer (STTT)*, 3(3):353–368, 2001. 5.3

[28] A. Nouri, S. Bensalem, M. Bozga, B. Delahaye, C. Jegourel, and A. Legay. Statistical model checking QoS properties of systems with SBIP. *Int. J. Softw. Tools Technol. Transf. (STTT)*, 17(2):171–185, April 2015. 1, 3.1

[29] A. Nouri, M. Bozga, A. Molnos, A. Legay, and S. Bensalem. Astrolabe: A rigorous approach for system-level performance modeling and analysis. *ACM Trans. Embed. Comput. Syst.*, 15(2):31:1–31:26, Mar. 2016. 6

[30] A. Nouri, B. L. Mediouni, M. Bozga, J. Combaz, A. Legay, and S. Bensalem. Performance evaluation of stochastic real-time systems with the sbip framework. Technical Report TR-2017-6, Verimag Research Report, 2017. 3.1, 4.1, 4.1, 6

[31] K. Sen, M. Viswanathan, and G. A. Agha. Vesta: A statistical model-checker and analyzer for probabilistic systems. In *International Conference on the Quantitative Evaluation of Systems, QEST'05*, pages 251–252, 2005. 6

[32] M. H. ter Beek, A. Legay, A. Lluch-Lafuente, and A. Vandin. Statistical analysis of probabilistic models of software product lines with quantitative constraints. In *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*, pages 11–15, 2015. 1

[33] A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945. 4.3

[34] H. L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon, 2005. 4.3

[35] H. L. S. Younes. Ymer: A statistical model checker. In *COMPUTER AIDED VERIFICATION, CAV'05*, pages 429–433. Springer, 2005. 1, 6