Integrating Simulink Models into the Model Checker Cosmos

Benoît Barbot, Béatrice Bérard, Yann Duplouy, Serge Haddad

Integrating Simulink Models into the Model Checker Cosmos *

Benoît Barbot[1], Béatrice Bérard[2], Yann Duplouy[3,4], and Serge Haddad[4]**

[1] LACL, Université Paris-Est Créteil
[2] Sorbonne Université, LIP6, CNRS UMR 7606, Paris, France
[3] IRT SystemX, Paris-Saclay, France
[4] LSV, ENS Paris-Saclay, CNRS, Inria, Université Paris-Saclay

(Tool paper - http://cosmos.lacl.fr/)

Abstract. We present an implementation for Simulink model executions in the statistical model-checker Cosmos. We take profit of this implementation for an hybrid modeling combining Petri nets and Simulink models.

Keywords: Performance Evaluation, Hybrid Systems, Statistical Model Checking, Simulink

Introduction

The validation of safety properties is a crucial concern for the design of computer guided systems, in particular for cyber-physical systems. For instance, in transport systems, a classical approach consists in analyzing the interactions of a randomized environment (roads, cross-sections, etc.) with a vehicle controller, often derived from a Simulink® model. However, while largely used in practice by industrial system designers, Simulink does not benefit from a formal semantics. Thus engineers usually have to infer the behaviours of their models from experiments which weakens the validation process. For this reason, several works proposed formal translations from (subsets of) Simulink blocks to other models like hybrid automata [14, 1], or languages like Lustre [15]. Other works directly define exact semantics [9] or operational semantics [10] for Simulink. We follow the latter approach and propose semantics for a significant fragment of Simulink. We proceed in two steps: we first develop an exact version, and then enrich it with effective procedures with the aim to efficiently integrate it into the statistical model checker Cosmos. With the resulting tool, we can obtain performance indices for models combining a randomized environment described by a stochastic Petri net and a controller described in Simulink, as illustrated for a double heater system.

* This research work has been carried out in the framework of IRT SystemX, Paris-Saclay, France, and therefore granted with public funds within the scope of the French Program "Investissements d'Avenir".
** The work of this author is supported by the project ERC EQualIS (FP7-308087).

1 The tool Cosmos

In [2], HASL an expressive temporal logic was introduced in order to analyze stochastic and hybrid discrete event systems. Its formulas are described by a Linear Hybrid Automaton (LHA) and an expression involving state variables and using path and stochastic operators. This logic is supported by a tool named Cosmos which performs statistical model checking of (ordinary or colored) stochastic Petri nets with general distributions. The main algorithm of this tool randomly simulates the net according to its stochastic semantics and synchronizes it with the execution of the formula's automaton. During the synchronization, it evaluates the expression of the formula providing a numerical (or boolean) value per trajectory. A statistical procedure decides when to stop the simulation and produces a confidence interval for the HASL expression.

The tool Cosmos consists of about 22000 lines of C++ code and is freely available at [4] under GPLv3. It relies on code generation to perform efficient simulation. It is divided into three main parts:

1. The parsing and code generation part reads the input files and the command line to build data structures for the net and the automaton. Then optimised C++ code simulating both the synchronized behaviour of the net and the automaton is generated.

2. The simulator library implements the core algorithm for synchronization and the generation of events using a pseudo random number generator.

3. The server part launches several copies of the simulator and aggregates their results. According to statistical parameters, a procedure decides whether enough trajectories have been simulated and stops all simulators when needed. Then, HASL expressions are evaluated.

Cosmos includes a family of statistical methods depending on the nature of the formula and the assumptions on the model. If the formula is boolean and corresponds to a comparison between the probability of an event and a threshold, it applies *sequential* hypothesis testing [16]. If the formula is numerical, it applies either a sequential method based on the Chow-Robbins bounds [11] or several *static* methods i.e. with a fixed number of trajectories. More precisely, when the formula returns a probability (corresponding to a Bernoulli distribution), it uses Clopper-Pearson bounds [12]. When the formula returns the expectation of a bounded random variable, it uses the Chernoff-Hoeffding bounds [13]. In the general case, it is based on the standard Gaussian approximation.

Cosmos has been successfullly applied in the context of flexible manufacturing systems [2] and biological networks [3, 6]. In addition, it has also been customised in order to address the challenges raised by different projects: simulation of rare events [7], cosimulation of a pacemaker software with a model of the human heart [8], sampling uniform trajectory for timed automata [5].

2 The tool Simulink

Simulink®[5] is a graphical programming environment for modeling and simulating dynamic systems. The main interface is a graphical block diagramming tool, with customizable libraries. It is used, for example, in the development of embedded systems for autonomous vehicles.

2.1 Syntax

We first introduce a formal syntax for the Simulink block diagrams, that we call *SK-models*. The set of types used in *SK*-models is denoted by $\mathcal{T}ype$ and contains booleans, integers and floating numbers, all of them seen (with a slight abuse) as subsets of the set $\mathbb{R}$ of real numbers. The time domain, denoted by $\mathbb{T}$, is an interval $[t_{\text{init}}, t_{\text{end}}]$ of $\mathbb{R}$.

A block contains a set of operators generating output signals from input signals.

Definition 1. *A* signal *of type $\mathcal{T}p \in \mathcal{T}ype$ is a function $s : \mathbb{T} \to \mathcal{T}p$, right-continuous, piece-wise $\mathcal{C}^\infty$, admitting a left limit $s(t^-)$ at each $t \in \mathbb{T}$. The set of signals of type $\mathcal{T}p$ defined on $\mathbb{T}$ is denoted by $\mathcal{S}ig_{\mathbb{T}}(\mathcal{T}p)$.*

An operator *is a function* $\text{op} : \mathcal{S}ig_{\mathbb{T}}(\mathcal{T}p_1) \times \ldots \times \mathcal{S}ig_{\mathbb{T}}(\mathcal{T}p_m) \to \mathcal{S}ig_{\mathbb{T}}(\mathcal{T}p)$ *such that for each $t \in \mathbb{T}$, the value $\text{op}(s_1, \ldots, s_m)(t)$ only depends on the restriction of the signals s_i on $[t_{\text{init}}, t]$.*

Blocks are classified according to three criteria:
(i) Whether they are continuously evaluated or sampled, in which case the block is said *discrete* and a *sampling delay* must be provided;
(ii) Whether there is a *latency* for evaluation of inputs, and whether this latency is *infinitesimal* or not. A latency is infinitesimal if it is either null (the block is said *immediate*) or such that the output value at time t only depends on the inputs on $[t_{\text{init}}, t[$ (like in integration). A non infinitesimal latency is said *positive*. A *non null* latency is a positive or infinitesimal non null latency.
(iii) Whether the output value depends on threshold crossing by an input signal, called *critical input* and denoted by i_c. In this case, the threshold values $(v_i)_{i \in I}$ must be specified, as a countable increasing sequence without accumulation point.

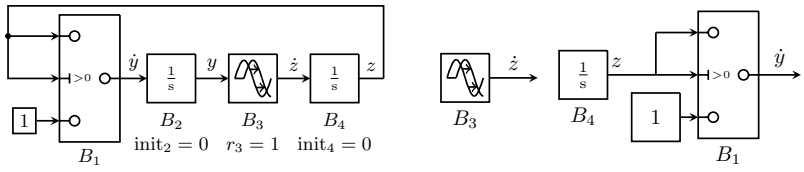
Definition 2 (Block type). *A* block type *is a tuple* $\mathsf{BT} = (n, m, (\text{op}_i)_{1 \le i \le n}, b_c, b_l, b_i, b_s, \text{Param})$ *where:*

- *n and m are the numbers of output and input signals respectively;*
- *$(\text{op}_i)_{1 \le i \le n}$ is a tuple of operators, one for each output signal;*
- *b_c, b_l, b_i, b_s are booleans indicating respectively if the block is discrete or continuous, immediate or with a latency, whether the latency is infinitesimal or not, and if the block is a threshold block;*

[5] https://fr.mathworks.com/products/simulink.html

– Param *is a set of parameters, including a sampling delay δ for a discrete block and a value r for a positive latency.*

An *SK*-model defines an architecture where blocks are instances of block types with specified parameters. The example of Fig. 1(a) features: Switch block



(a) An example of *SK*-model (b) Backward graphs

Fig. 1. An example of *SK*-model and its backward graphs decompositions

B_1 (continuous, immediate, with null latency and a single threshold) with operator: $\mathrm{op}(i_1, i_c, i_2)(t) = \mathsf{if}(i_c(t) > 0, i_1(t), i_2(t))$; Integrator block B_2, with operator defined by $\mathrm{op}(i)(t) = \int_{t_{\mathrm{init}}}^{t} i(\tau)\mathrm{d}\tau$. This is a continuous block with infinitesimal latency; a *Transport Delay* (continuous) block B_3 which outputs its input signal with a latency $r = 1$.

Definition 3 (*SK*-Model). *An SK-model $\mathcal{M} = (\mathcal{B}, L)$ consists of:*
(1) A set $\mathcal{B}$ of blocks, defined by their respective type and parameter values. We denote by $\langle B, o\rangle$ (resp. $\langle i, B\rangle$) an output o (resp. an input i) of block B.
(2) A set L of links *of the form $(\langle B', o\rangle, \langle i, B\rangle)$ satisfying: for any input $\langle i, B\rangle$, there is exactly one output $\langle B', o\rangle$ such that $(\langle B', o\rangle, \langle i, B\rangle) \in L$.*

In the sequel, we restrict the definition above to so-called *correct SK*-models, by forbidding the presence of immediate cycles: An immediate cycle is a sequence of links $(\langle B_1, o_1\rangle, \langle i_2, B_2\rangle), (\langle B_2, o_2\rangle, \langle i_3, B_3\rangle), \ldots (\langle B_{n-1}, o_{n-1}\rangle, \langle i_n, B_n\rangle)$ such that $B_n = B_1$ and for each i, $1 \leq i \leq n$, B_i is an immediate block.

The correctness of an *SK*-model can be checked in linear time by a topological sort on an associated graph restricted to immediate blocks.

2.2 Exact and approximate semantics

Simulink® models represent hybrid systems, combining discrete and continuous components. The *trajectory* of an *SK*-model $\mathcal{M}$, if it exists, is the vector $\boldsymbol{w}$ of all values of output signals over $\mathbb{T}$. Signal evaluation requires to split the interval $[t_{\mathrm{init}}, t_{\mathrm{end}}]$ into a finite sequence of contiguous sub-intervals, on which the trajectory is the solution of a system of differential equations. Discrete samplings and threshold crossings are located at the boundaries of these sub-intervals.

Given the set Δ of sampling delays of $\mathcal{M}$, the set R of positive latencies, a default sampling delay $\delta_{\max}$, and $\delta_{\mathrm{lat}} = \min(\delta_{\max}, \min(R))$, a first static value for the next interval bound after t is defined by $\mathrm{next}_s(t) = \min(\delta_{\mathrm{samp}}(t), t+\delta_{\mathrm{lat}}, t_{\mathrm{end}})$,

where $\delta_{\text{samp}}(t) = \min\{p\delta \mid \delta \in \Delta, p \in \mathbb{N}, p\delta > t\}$. The value $\text{next}_s(t)$ may then be decreased to take the crossings into account.

Differential equations. For an *SK*-model $\mathcal{M}$, we denote by $Th(\mathcal{M})$ the set of threshold blocks and by $Lat(\mathcal{M})$ the set of blocks with positive latency. A *mode* is the choice, for each block in $Th(\mathcal{M})$, of an interval between thresholds (according to the sequence $(v_i)_{i\in I}$ of this block) and an *environment* is the choice, for each block in $Lat(\mathcal{M})$, of an output signal. Given a mode m and an environment ℓ, the set of differential equations is obtained by a backward exploration: For an integrator block B, this exploration starts from the block B^- for which the output o^- is the input of B. *Terminal blocks* are those for which the output can directly be obtained from previous values: (i) blocks without input, (ii) integration blocks and (iii) blocks with positive latency. For the *SK*-model of Fig. 1a, some backward graphs resulting from this exploration are depicted in Fig. 1b. This backward exploration yields a differential equation $\dot{\boldsymbol{x}} = F_{m,\ell}(\boldsymbol{x})$ of which $\boldsymbol{v}$, the sub-vector of $\boldsymbol{w}$ containing the output signals of integrator blocks could be a solution.

For instance, in the *SK*-model of Fig. 1a over $\mathbb{T} = [0, 2]$, the initial values are $t_0 = 0$, $y(0) = 0$, and $z(0) = 0$, with $m_0^{(1)} =]-\infty, 0]$ and $\dot{z}(t) = 0$ on $[0, 1]$ where the exponent (j) denotes block B_j. Choosing $t_1 = 1$, the differential equations are $\dot{y} = 1$ and $\dot{z} = 0$, which yields $z(t) = 0$ and $y(t) = t$ over $[0, 1[$ with mode specified by $m_0^{(1)} =]-\infty, 0]$ and latency $\ell_0^{(3)}(t) = 0$.

Existence of a trajectory. The vector $\boldsymbol{w}$ is a trajectory of the model $\mathcal{M}$ over $\mathbb{T} = [t_{\text{init}}, t_{\text{end}}]$ if there exists (i) a sequence $(t_i)_{i\in[\![0,N]\!]}$ with $t_0 = t_{\text{init}}$ and $t_N = t_{\text{end}}$, (ii) a minimal sampling delay $\varepsilon_{\mathbb{T}} > 0$, which will be a lower bound on the length of sub-intervals, and (iii) a sequence $(m_i, \ell_i)_{i\in[\![0,N-1]\!]}$ of modes and environments, such that:

0. The initial mode m_0 and environment ℓ_0 are those of the initial values of the model. Moreover, for $0 < i < N$, ℓ_i on $[t_i, t_{i+1}[$ agrees with $\boldsymbol{w}$ on $[t_0, t_i[$.

1. For all $i < N$, $\dot{\boldsymbol{x}} = F_{m_i,\ell_i}(\boldsymbol{x})$ admits a solution on the interval $[t_i, t_{i+1}+\varepsilon_{\mathbb{T}}]$ which coincides with $\boldsymbol{v}$ on $[t_i, t_{i+1}[$ and is consistent with m_i on $]t_i, t_{i+1}[$. The vector $\boldsymbol{w}$ agrees with the operations of the remaining blocks.

2. For all $0 \le i < N - 1$, $\boldsymbol{w}(t_{i+1})$ corresponds to the application to $\boldsymbol{w}(t_{i+1}^-)$ of the operators of *active* discrete blocks (i.e. such that t_{i+1} is a sampling time). The solution of $\dot{\boldsymbol{x}} = F_{m_i,\ell_{i+1}}(\boldsymbol{x})$ on $[t_{i+1}, t_{i+1} + \varepsilon_{\mathbb{T}}]$ is consistent with m_{i+1}.

Returning to the example of Fig. 1, the environment is $\ell_1^{(3)}(t) = y(t-1) = t-1$ on $[1, 2]$, and the mode $m_1^{(1)} =]0, +\infty[$ must agree with the value of z at time $1 + \varepsilon_{\mathbb{T}}$. Hence with $t_2 = 2$, the differential equations are $\dot{z}(t) = t - 1$ and $\dot{y} = z$, which yields $z(t) = (t-1)^2/2$ and $y(t) = (t-1)^3/6 + 1$.

There are several cases where a trajectory does not exist: for instance if some differential equation (like $\dot{y} = 2^y$) cannot be solved on the interval, or if the solution violates the mode constraints related to some threshold block. With suitable hypotheses on the operators, we prove that if a trajectory exists, it is unique. Unfortunately, as is often the case for hybrid systems, the existence of a trajectory is an undecidable problem.

Approximate semantics. Since the resolution of differential equations and the determination of threshold crossings are not effective operations, an approximate version of the semantics above is needed for implementation in the tool Cosmos.

The search for a trajectory relies on an iterative construction of a partition into sub-intervals $[t_{\text{init}}, t_{\text{end}}] = \bigcup_{i=0}^{N-1} [t_i, t_{i+1}]$. We emphasize three main features:
(1) The signal values are only stored at times $(t_i)_{0 \leq i \leq N}$: For each output signal o of a block B, an array $W_{B,o}[i]$ of its values is kept for each time t_i. This implies interpolation operations to compute signal values at intermediate times.
(2) When $t_0, \ldots, t_i$ are built, the static next step value $\text{next}_s(t_i)$ must be refined to take into account variable integration steps, as done in the Runge-Kutta-Fehlberg (or ODE45) method used here. Here, we omit the adaptation details of these classical procedures, performed with the constant mode of t_i, which produce the new evaluating time t_{i+1} and the associated values.
(3) In addition to $\varepsilon_{\mathbb{T}}$, we introduce ε_V, the minimal discerning capacity of the 'controller' (i.e. $|x| < \varepsilon_V \Rightarrow x \simeq 0$), to handle the termination tests in the adaptative integration method (like ODE45).

Besides the implementation objective, the main purpose of this approximate semantics is to find suitable hypotheses ensuring that if a trajectory exist w.r.t. exact semantics, then there exists a close approximate one: for any $\varepsilon > 0$ there exist $\varepsilon_{\mathbb{T}}$ and ε_V such that for each output o of block B and for all times t_i, $|W_{B,o}[i] - w_{B,o}(t_i)| < \varepsilon$.

3 Integrating Simulink into Cosmos

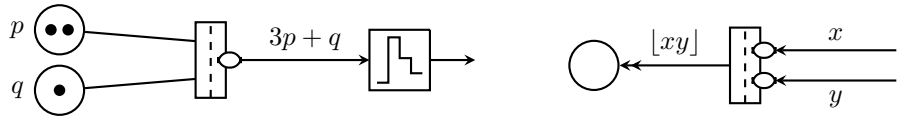
In order to generate trajectories, Cosmos maintains a queue of activated events (transition firings in the net or state changes in the HASL automaton) selecting the earliest one. The integration of an *SK*-model is managed through the specification of a new event, called *SK*-event, corresponding to the next sampling time of the *SK*-model. This requires to specify how the models interact, implying transformation of discrete values (like the number of tokens in places) into continuous ones (like signal values), and vice-versa. We have chosen to perform these operations via special transitions called *interface transitions*.

Interface transitions. There are two kinds of interface transitions: *SK*-in transitions, directed from the net to the *SK*-model and *SK*-out transitions in the other direction.

The input arcs of a *SK*-in transition (see Fig. 2a) are *test arcs* (explained with the firing) connected to places of the net. Any output arc is connected to the input of a Simulink block. An *SK*-in is enabled when the content of at least one of its input place is modified. The firing of such a transition proceeds as follows: a function is associated with each output arc, taking as parameters the contents of the input places. When the firing takes place, the function is evaluated. This function can be specified by a multiset of tokens as illustrated, or by a C code associated with the arcs.

The input arcs of an *SK*-out transition (see Fig. 2b) are output signals of an *SK*-model and the output arcs are *overwriting* arcs connected to places that

can only be connected to ordinary transitions of the net by *read* arcs. Similarly to *SK*-in transitions, the output arcs are labelled by functions of the incoming signals. Such a transition is activated at every sampling time of the *SK*-model. Upon firing, it rewrites the contents of the output places according to the evaluation of the function.



(a) An *SK*-in transition (b) An *SK*-out transition

Fig. 2. Petri net/Simulink Interface transitions

Simulation loop. We now describe in more details the simulation loop, which enhances the standard Cosmos simulation loop. All enabled transitions are stored into an *event* queue implemented as a binary heap, with their time of occurrence, their priority and weight. The next Simulink step is added as a possible event. At each simulation step, the earliest event is chosen. Among simultaneous events, the (decreasing) priority order is the following: *1. SK*-out firings, *2.* ordinary transition firings, *3. SK*-in firings, *4. SK*-event. In case of equal priorities, the choice is randomized according to the weights. Once an event is selected:

- If it is an ordinary transition firing, the marking is updated, the associated C code is executed; transitions that are newly enabled trigger new events while events corresponding to disabled transitions are removed.
- If it is an *SK*-in firing, the Simulink signals are updated and the time of the Simulink event is set to the current time.
- if it is the *SK*-event, all output signals are updated and the time of the Simulink event is updated as presented in section 2.2. Finally, the *SK*-out transitions are added to the event queue with the current time.
- if it is a *SK*-out firing, the contents of output places are updated.

To simulate a discrete event system, at each step, one only has to compute what the next event will be and increase the simulation time to the time of this event. This is how ordinary Petri net transitions are fired in Cosmos. This leads to an efficient simulation of such system as the time to compute a simulation depends on the number of events and not on the simulated time. Unfortunately, this property is lost when simulating hybrid systems: the *SK*-event is triggered at least at a fixed frequency (δ_{max}). In the next section we experimentally study the impact of integration on simulation time.

4 Benchmarks

Among the multiple systems that can be modeled within this framework, we choose a well known toy (but still relevant) example: a device with two heaters

prone to faults and using bang-bang controllers to keep the temperature in a room between $20°C$ and $25°C$. The system is modeled by a stochastic Petri net (Fig. 3) with randomized faults and repairs. The evolution of room temperature and heater behaviours are hybrid and thus are modeled in Simulink (Fig. 4). The fault transitions of the net have an exponential time distribution (with different rates). The repairman, initially at the Idle state, randomly chooses which (faulty) heater he will repair, then proceeds in fixed time and goes back to the Idle state. By default, both heaters are working (places Op_1 and Op_2 have a token).

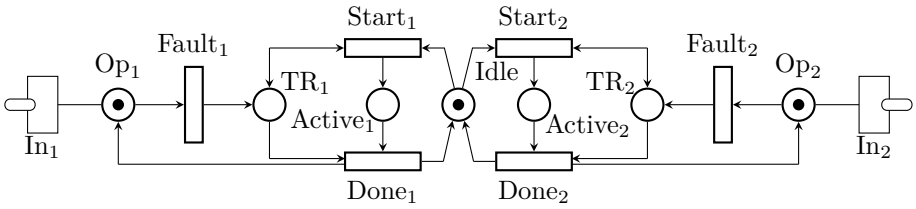


Fig. 3. Petri net handling faults and repairs of a double heater

The Simulink model handles the differential equations for both heaters, and for the outside temperature which is modelled by a sine wave (T_{ext}). The differential equation is : $\dot{T} = \mathbb{1}_{On_1} c_1 (T_{h_1} - T) + \mathbb{1}_{On_2} c_2 (T_{h_2} - T) + c_{ext}(T_{ext} - T)$ where c_1, c_2, and c_{ext} are the respective thermal conductivity coefficients, T_{h_1} and T_{h_2} are the respective temperatures at which each heater functions, and On_1 and On_2 are the respective states of each bang-bang controller which should maintain the temperature between $T_{min} = 20°C$ and $T_{max} = 25°C$. A bang-bang controller is a very simple hysteresis controller where the heater is switched on ($On_i = 1$) when the temperature decreases to T_{min} and switched off ($On_i = 0$) when the temperature increases to T_{max}. The inputs $F1$ and $F2$ receive respectively the content of places Op_1 and Op_2.

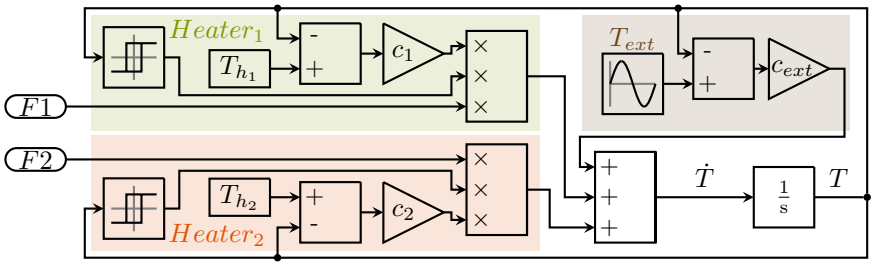


Fig. 4. A Simulink model computing differential equations for the double heater. The model contains four parts: the two heater temperatures, the external temperature and the block completing the differential equation

Figure 5 shows a simulation of the system. In the first period there is only a small failure of heater 2, and we can observe the bang-bang behaviours of the system. In the second period both heaters fail at the same time while the outside temperature is low, thus the temperature quickly drops to $13°C$ before the first heater is repaired.

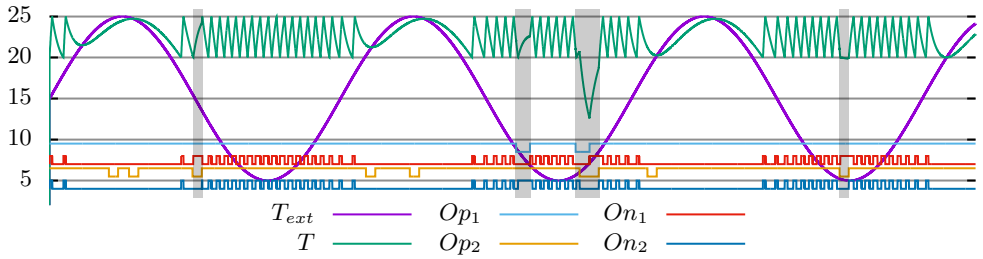


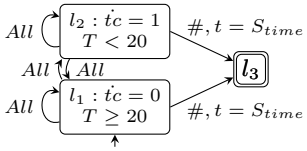
Fig. 5. A trace of simulation: T and T_{ext} represent the inside and outside temperatures, Op_i corresponds to heater i being operational and On_i corresponds to heater i being switched on. Gray areas highlight failure of at least one heater when $T_{ext} < T_{min}$.

We are interested in several performance indices. The first type concerns the reliability of the model measured by two indices: *the minimal temperature observed along a trajectory* (I_1) and *the time spent in a state where the temperature is below* $20°C$ (I_2). The second type concerns the average behaviour: *the average temperature* (I_3), *the average number of switched-on heaters* (I_4), which is correlated with the energy consumption of the system, and *the average time during which the repairman is idle* (I_5).

These indices are specified in HASL with an LHA (Fig. 6) which accepts the trajectories after S_{time} time units. The LHA contains a hybrid variable tc with derivative 1 when the temperature is below $20°C$ and 0 otherwise. The HASL expressions start with a *probability operator*: here AVG is used for all indices to specify the average value over all trajectories; then a *path operator* (Min, Last, Mean) which is defined along each path. Path operators take as parameters algebraic expressions over the Petri net places and the LHA variables. For example, I_1 specifies the minimal temperature along a trajectory and then the average value over all trajectories.

I_1 : AVG(Min(T))
I_2 : AVG(Last(tc))
I_3 : AVG(Mean(T))
I_4 : AVG(Mean($Active_1 + Active_2$))
I_5 : AVG(Mean($Idle$))

(a) HASL formulas



(b) LHA

Fig. 6. HASL specification for performance indices

The model as it is described above is referred to as M_0. In order to study the overhead of integral computations over the stochastic simulation, we build two additional alternative models. The first one M_1 is a model where the integration block in the Simulink diagram has been replaced by a discrete time integrator. In the model M_2, the simulink part is omitted, keeping only events that are transition firings.

Indices	M_0	M_1
I_1	[18.698 ; 18.716]	[18.272 ; 18.289]
I_2	[66.344 ; 67.217]	[92.921 ; 93.927]
I_3	[22.439 ; 22.442]	[22.485 ; 22.488]
I_4	[0.4999 ; 0.5006]	[0.4884 ; 0.4890]
I_5	[0.9239 ; 0.9242]	[0.9239 ; 0.9241]

Models	Build time	Sim. time
M_0	5.74s	6 885s
M_1	5.73s	1 145s
M_2	1.31s	1.810s

Table 1. Simulation results

Each model was run for 500 000 simulations of 2 000 seconds, with $\varepsilon_V = 0.01$ and $\delta_{max} = 1$. The sine wave frequency was 0.01 and oscillating between 5°C and 25°C, and the time step of the discrete-time integrator was δ_{max} (1 second). We used $T_{h_1} = 55$°C, $T_{h_2} = 65$°C, $c_1 = 0.02$, $c_2 = 0.013$ and $c_{ext} = 0.04$. Results are reported in Table 1. The left table reports the computed confidence interval for the different indices, the right one reports simulation and building times.

Tool analysis. The build time is always less than the simulation time and becomes negligible when models include a Simulink part. The critical factors for simulation time are: (i) the speed of step firing, about 10^{-7} sec. for net firing compared to 10^{-6} sec. for Simulink steps, and (ii) the number of steps per trajectory, about 40 for M_2 vs. 2000 for M_1. As expected, the use of a discrete-time Integrator yields a faster simulation, albeit still far longer than the net alone, while it affects the accuracy of the index values and more precisely triggers a larger variation of temperature over time.

Property analysis. We focus on the most pertinent model M_0, with two antagonist goals: minimizing the installation cost (depending on the parameters of heaters and repairman), and maximizing the comfort of the user (depending on the temperature evolution). With the current parameters, each heater is active about 1/4 of the time and the repairman is idle 92% of the time. The average temperature is about 22°C, reaching the objective, while the minimal temperature is slightly above 18°C.

References

1. Agrawal, A., Simon, G., Karsai, G.: Semantic Translation of Simulink/Stateflow Models to Hybrid Automata Using Graph Transformations. Electr. Notes Theor. Comput. Sci. 109, 43–56 (2004)
2. Ballarini, P., Barbot, B., Duflot, M., Haddad, S., Pekergin, N.: Hasl: A new approach for performance evaluation and model checking from concepts to experimentation. Performance Evaluation 90(0), 53 – 77 (2015)
3. Ballarini, P., Duflot, M.: Applications of an expressive statistical model checking approach to the analysis of genetic circuits. Theor. Comput. Sci. 599, 4–33 (2015)
4. Barbot, B., Ballarini, P., Djafri, H.: http://cosmos.lacl.fr
5. Barbot, B., Basset, N., Beunardeau, M., Kwiatkowska, M.: Uniform Sampling for Timed Automata with Application to Language Inclusion Measurement. In: Proceedings of QEST 2016. LNCS, vol. 9826, pp. 175–190. Springer (2016)
6. Barbot, B., Haddad, S., Heiner, M., Picaronny, C.: Rare event handling in signalling cascades. International Journal on Advances in Systems and Measurements 8(1-2), 69–79 (Jun 2015)
7. Barbot, B., Haddad, S., Picaronny, C.: Coupling and importance sampling for statistical model checking. In: Proceedings of TACAS'12. pp. 331–346 (2012)
8. Barbot, B., Kwiatkowska, M., Mereacre, A., Paoletti, N.: Building Power Consumption Models from Executable Timed I/O Automata Specifications. In: Proceedings of HSCC 2016. pp. 195–204. ACM (2016)
9. Benveniste, A., Bourke, T., Caillaud, B., Pouzet, M.: Non-standard semantics of hybrid systems modelers. Journal of Computer and System Sciences 78(3), 877 – 910 (2012)

10. Bouissou, O., Chapoutot, A.: An operational semantics for simulink's simulation engine. In: Proceedings of the 13th ACM SIGPLAN/SIGBED. pp. 129–138. LCTES '12, ACM, New York, NY, USA (2012)
11. Chow, Y.S., Robbins, H.: On the asymptotic theory of fixed-width sequential confidence intervals for the mean. The Annals of Math. Statistics pp. 457–462 (1965)
12. Clopper, C., Pearson, E.S.: The use of confidence or fiducial limits illustrated in the case of the binomial. Biometrika pp. 404–413 (1934)
13. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American statistical association 58(301), 13–30 (1963)
14. Tiwari, A.: Formal Semantics and Analysis methods for Simulink Stateflow Models. Tech. rep., SRI (2002)
15. Tripakis, S., Sofronis, C., Caspi, P., Curic, A.: Translating discrete-time Simulink to Lustre. ACM Trans. Embedded Comput. Syst. 4(4), 779–818 (2005)
16. Wald, A.: Sequential tests of statistical hypotheses. The Annals of Math. Statistics 16(2), 117–186 (06 1945)