Stochastic simulation of AltaRica 3.0 models

M. Batteux & A. Rauzy
LIX – École Polytechnique
Route de Saclay, Palaiseau, France

ABSTRACT: The aim of this article is to present the stochastic simulator of the AltaRica 3.0 project. Stochastic simulation is an important tool for safety and reliability analyses of systems. It provides fine results, even with complex systems, to calculate safety and reliability indicators. The principle is to run many pseudo-random simulations, corresponding to histories of the behavior of the system, and to make statistics on them.
The stochastic simulator of the AltaRica 3.0 project was designed by taking into account original features, not only by introducing concepts within the language, but also with its implementation. First, considered delay functions are a few built-in ones representing the most widely used in safety and reliability analyses (exponential, Weibull, etc.). For more specific ones, a generic mechanism allows users to define them point wisely by means of sets of points interpolated in a triangular way. In addition, observers are defined in a generic way. They are quantities to be observed during simulations in order to perform safety and reliability analyses. Finally, compilation techniques are used to reduce computation time of simulations because stochastic simulation needs an important number of simulations to stabilize the measures.
This work is a part of AltaRica 3.0 project, which aims to propose a set of authoring, simulation and assessment tools to perform Model-Based Safety Analyses. The new version of AltaRica modeling language is in the heart of the project. Its associated assessment tools already include Fault Trees or Markov Chains compilers, stochastic and stepper simulators, etc.

1 INTRODUCTION

1.1 *AltaRica in safety and reliability analyses*

The Model-Based approach for safety and reliability analyses is gradually wining the trust of engineers and is still an active domain of research. Safety engineers master "traditional" risk modeling formalisms, such as "Failure Mode, Effects and Criticality Analysis" (FMECA), Fault Trees (FT), Event Trees (ET), Markov Processes. Efficient algorithms and tools are available. However, despite of their qualities, these formalisms share a major drawback: models are far from the specifications of the systems under study. As a consequence, models are hard to design and to maintain throughout the life cycle of systems. A small change in the specifications may require revisiting completely safety models, which is both resource consuming and error prone.

The high-level modeling language AltaRica has been created to tackle this problem. AltaRica models are made of hierarchies of reusable components. Graphical representations are associated with components, making models visually very close to Process and Instrumentation Diagrams. An entirely new version of the language, AltaRica 3.0, is developed. Its underlying mathematical model – Guarded Transition Systems, first introduced in Rauzy (2008) and extended in Prosvirnova and Rauzy (2012) – makes it possible to design acausal components and to handle looped systems. The development of a complete set of freeware authoring and assessment tools is planed, so to make them available to a wide audience.

1.2 *Stochastic simulation in safety and reliability analyses*

Computer simulation is a computer program attempting to simulate an abstract model of a particular system (i.e.: make experiments), in order to understand its behavior or evaluate strategies to control it (Rubinstein & Melamed 1998). Simulate means build an history of the "behavior" of the system. Such history is called a simulation. This technique is simple to implement and allows to study systems constituted by several components. A stochastic simulation consists on randomly build an history: i.e. to make pseudo-random choices when several options are available during the simulation.

Within safety and reliability domains, stochastic simulation is an important tool to evaluate performances of systems (Cancela and Khadiri 1998, Baca

1993). It is based on Monte Carlo simulation (Coates et al. 1988, Dubi 2000, Liu 2008) which the main principle is to run several such pseudo-random simulations and to make statistics on them.

1.3 *Stochastic simulation with AltaRica 3.0*

This paper presents the stochastic simulator of the AltaRica 3.0 project. It was designed by taking into account original features, not only by introducing some concepts within the language, but also in its implementation.

We have first consider delay functions in a generic manner. In fact, a few built-in delay functions are implemented by the simulator. They represent the most widely used in safety and reliability analyses (e.g.: Dirac, exponential, Weibull, etc.). For more specific ones, a generic mechanism allows users to define them point wisely. They are described by means of sets of points interpolated in a triangular way.

We have then defined observers also in a generic way. Observers are indicators to be observed in order to perform safety and reliability analyses. They correspond to quantities calculated during simulations. They can be defined in a easy way and two types are considered: observers on events, to observe average numbers of times events have been fired and fireable; observers on formulas, to observe average numbers of times different values have been taken by formulas.

Finally we have used compilation techniques to reduce computation time of simulations. In fact, stochastic simulation needs an important number of simulations to stabilize the measures. With nowadays computers, perform up to several millions of histories (of reasonable length) is relatively easy. Thus reducing the computation time of simulations gets an issue and with that respect, compilation techniques may be of a great help.

The remainder of this article is organized as follows. In the second part, we introduce the overview of the AltaRica 3.0 project with the new version of the modeling language. In the third part, we present the pivot formalism of Guarded Transition System, to compile AltaRica 3.0 models, taken into account by the stochastic simulator. Original features are then presented in the next three parts. The fourth part presents delay functions, the fifth part presents observers and the sixth part presents compilation techniques. Finally the last part concludes by summarizing the design and implementation of the stochastic simulator and outlines interesting directions for future works.

2 ALTARICA 3.0

2.1 *The AltaRica 3.0 project*

The objective of the AltaRica 3.0 project is to propose a set of modeling and assessment tools to perform safety analyses.

The new version of modeling language AltaRica 3.0 is in the heart of this project. It significantly increases the expressive power of the previous version without decreasing its algorithmic efficiency. AltaRica 3.0 models are compiled into a low level formalism: Guarded Transition Systems. Guarded Transition Systems is a state/transition formalism that generalizes classical safety formalisms, such as Reliability Block Diagrams, Petri Nets and Markov chains.

The assessment tools, for Guarded Transition Systems, include a Fault Tree compiler to perform Fault Tree Analysis (FTA), a Markov chain generator, stochastic and stepwise simulators, a model-checker, and a reliability allocation module. These tools will be distributed under a free license. The assessment tools enable users to perform virtual experiments on systems, via models, to compute different kinds of reliability indicators and, also, to perform cross check calculations. Thanks to these tools AltaRica models can be used to perform Preliminary System Safety Analysis (PSSA) and System Safety Analysis (SSA).

2.2 *AltaRica 3.0 Modeling Language*

The previous version of AltaRica modeling language, AltaRica Data-flow, is a generalization of both Petri nets and block diagrams. From the former, it has imported the notions of states, events and guarded transitions whereas the latter inspired the notions of events synchronization, hierarchical description and flows. This last notion makes it possible to represent remote interactions in a simple way. However, located synchronizations cannot be captured and bidirectional flows circulating through a network cannot be modeled in a natural way. Moreover, it remains difficult to model looped systems. For these reasons AltaRica Data-Flow is not powerful enough to model such complex systems.

Thus, a new version of the language, so-called AltaRica 3.0, is currently under definition. It improves the previous version AltaRica Data-Flow into two directions. Its semantic is based on the new underlying mathematical model: Guarded Transitions Systems. It provides new constructs to structure models. The new underlying formalism makes it possible to handle systems with instant loops and to define acausal components (components for which the input and output flows are decided at run time). It is much easier to model systems with bidirectional flows (e.g. electrical systems).

3 GUARDED TRANSITION SYSTEMS

First introduced in Rauzy (2008), Guarded Transition Systems is a pivot formalism for Safety modeling and analyses. It generalizes classical formalisms, such as Reliability Block Diagrams, Markov chains and Petri Nets. The new semantics of instructions, presented in

Prosvirnova & Rauzy (2012), makes it possible to represent components with bidirectional flows.

3.1 *Definition*

A Guarded Transition Systems, noted GTS, is a quintuple $\langle V, E, T, A, \iota \rangle$, where:

- $V = S \uplus F$ is a set of variables, divided into two disjoint sets S of state variables and F of flow variables.
- E is a set of symbols, called events.
- T is a set of transitions denoted $e : G \to P$.
- A is an assertion, i.e. an instruction built over V.
- ι is an assignment of variables of V, so-called an initial or default assignment.

A transition $e : G \to P$ is a triple $\langle e, G, P \rangle$ where $e \in E$ is an event, G is a guard (i.e.: a boolean formula built over V) and P is an instruction built over V, called an action or a post-condition. A transition $e : G \to P$ is said *fireable* in a given state σ (i.e. for a given variable assignment σ) if its guard G is satisfied in this state.

3.2 *Instructions*

Both assertions and actions of transitions are described by means of instructions. An instructions can be:

- The empty instruction $skip$.
- An assignment $v := E$, where v is a variable and E is an expression built over variables from V.
- a conditional assignment $if\ C\ then\ I$, where C is a Boolean expression and I is an instruction.
- a block $\{I_1, \ldots, I_n\}$ of instructions.

State variables can occur as the left member of an assignment only in the action of a transition; whereas flow variables can only in the assertion. Instructions are interpreted in a slightly different way depending they are used in the actions or in the assertion. Let σ be the variable assignment before the firing of the transition $e : G \to P$. Applying the instruction P to the variable assignment σ consists in calculating a new variable assignment τ as follows. We start with $\tau = \sigma$. Then,

- if P is an empty instruction, then τ is left unchanged.
- if P is an assignment $v := E$, then $\tau(v)$ is set to $\sigma(E)$. If the value of v has been already modified and is different from the calculated one, then an error is raised.
- if P is a conditional assignment $if\ C\ then\ I$ and $\sigma(C)$ is true, then the instruction I is applied to τ.
- if P is a block of instructions $\{I_1, \ldots, I_n\}$ then instructions I_1, …, I_n are successively applied to τ.

It is important to note that in the above mechanism, right hand side of assignments and conditional expressions are evaluated in the context of σ. Thus, the result does not depend on the order in which instructions of a block are applied. In other words, instructions of a block are applied in parallel. Let denote by $Update(P, \sigma)$ the variable assignment τ resulting from the application of the instruction P to σ.

Let A be the assertion and let τ be the variable assignment obtained after the application of the action of a transition. Applying A consists in calculating a new variable assignment (of flow variables) π as follows. We start by setting all state variables in π to their values in τ: $\forall v \in S \pi(v) = \tau(v)$. Let D be a set of unevaluated flow variables, we start with $D = F$. Then,

- if A is an empty instruction, then π is left unchanged.
- if A is an assignment $v := E$, then if $\pi(E)$ can be evaluated in π, i.e. all variables of E have a value in π, then $\pi(v)$ is set to $\pi(E)$ and v is removed from D. If the value of v has been already modified and is different from the calculated one, then an error is raised.
- if A is a conditional assignment $if\ C\ then\ I$ and $\pi(C)$ can be evaluated in π and $\pi(C)$ is true, then the instruction I is applied to π. Otherwise, π is left unchanged.
- if A is a block of instructions $\{I_1, \ldots, I_n\}$ then instructions I_1, …, I_n are repeatedly applied to π until there is no more possibility to assign a flow variable.

If after applying A to π there are unevaluated variables in D, then all these variables are set to their default values $\forall v \in D \pi(v) = reset(v)$ and A is applied to π in order to verify that all assignments are satisfied. If that is not true an error is raised. Let denote by $Propagate(A, \sigma)$ the variable assignment resulting from the application of the instruction A to σ.

3.3 *Reachability graph*

Assume that σ is the variable assignment just before the firing of a transition. Then, the firing of the transition transforms σ into the assignment $Fire(e : G \to P, A, \sigma)$ defined as follows:

$$Fire(e : G \to P, A, \sigma) = Propagate(A, Update(P, \sigma))$$

Guarded Transition Systems are implicit representations of Kripke structures, i.e. of graphs whose nodes are labeled by variable assignments and whose edges are labeled by events. More exactly, the semantics of a Guarded Transition Systems $\langle V = S \uplus F, E, T, A, \iota \rangle$ is a Kripke structure, i.e. a graph $\Gamma = (\Sigma, \Theta)$, where Σ is a set of variable assignments (also called states and representing nodes of the graph) and Θ is a set of triples $\langle s, e, q \rangle$, $s, q \in \Sigma$, $e \in E$ (representing transitions of the graph). Γ is the smallest Kripke structure, such that the following is verified:

1. σ_0 is the initial state of the Kripke structure: i.e. $\sigma_0 = Propagate(A, \iota, \iota) \in \Sigma$.
2. if $\sigma \in \Sigma$ and $\exists t = \langle e, G, P \rangle \in T$, such that $\sigma(G) = TRUE$, then the state $\tau = Fire(P, A, \iota, \sigma)$ is in Σ and the transition (σ, e, τ) is in Θ,

The calculation of $\Gamma = (\Sigma, \Theta)$ may raise errors. A well designed Guarded Transition Systems avoids this problem. The Kripke structure Γ is also called a reachability graph.

3.4 *Timed/Stochastic Guarded Transition Systems*

A probabilistic time structure can be put on top of a Guarded Transition System so to get timed/stochastic models. The idea is to associate to each event the following information:

- A delay that can be deterministic or stochastic and may depend on the state. When a transition labeled with the event becomes fireable at time t, a delay d is calculated and the transition is actually fired at time $t + d$ if it stays fireable from t to $t + d$.
- A weight, so called expectation, which is used to determine the probability that the transition is fired in case several transitions are fireable at the same date.

3.4.1 *Timed Guarded Transition Systems*

Formally, a Timed Guarded Transition System is a tuple $\langle V, E, T, A, \iota, delay \rangle$, where $\langle V = S \uplus F, E, T, A, \iota \rangle$ is a Guarded Transition System and $delay : E \to \mathbb{R}_+$ is a function, that associates to each event a non-negative real number.

The semantics of Timed Guarded Transition Systems can be defined in terms of executions. An execution is a sequence $(\sigma_0, d_0, t_0) \stackrel{e_0}{\to} (\sigma_1, d_1, t_1) \ldots (\sigma_{n-1}, d_{n-1}, t_{n-1}) \stackrel{e_{n-1}}{\to} (\sigma_n, d_n, t_n)$, where $t_i \in \mathbb{R}_+$ represents the date when the event e_i occurs and the transition labeled by this event is fired. The step $(\sigma_i, d_i, t_i) \stackrel{e_i}{\to} (\sigma_{i+1}, d_{i+1}, t_{i+1})$ corresponds to the advancement of time and to the firing of the transition labeled by the event e.

3.4.2 *Stochastic Guarded Transition Systems*

The timed interpretation of GTS does not specify how delays are calculated. Therefore, it encompasses the case where delays are stochastic. It remains however to define what stochastic means. To do so, we shall introduce the notion of oracle. The idea is to delegate all the "randomness" of a run to an oracle. In this way, the GTS stays purely deterministic but its behavior depends on the outcomes of the oracle. Oracles concentrate therefore the non-determinism of executions.

More formally, an *oracle* o is an infinite sequence of real numbers comprised between 0 and 1 (included): i.e. $o : \mathbb{N} \to [0; 1] \subset \mathbb{R}$. The only operation available on an oracle is to consume its first element. This operation returns the first element and the remaining of the sequence, which is itself an oracle.

Finally, a Stochastic Guarded Transition System is a tuple $\langle V = S \cup F, E, T, A, \iota, delay, expectation \rangle$, where:

- $\langle V = S \uplus F, E, T, A, \iota \rangle$ is a GTS;
- $delay$ is a function from events and oracles to non-negative real numbers.

When several transitions are scheduled to be fired at the same date, one is picked at random by using the oracle and according to their expectations. The probability $p(e_k : G_k \to P_k)$ to fire the transition $e_k : G_k \to P_k$, is defined as follows:

$$p(e_k : G_k \to P_k) = \frac{expectation(e_k)}{\sum_{e_i : d_n(e_i) = 0} expectation(e_i)} \qquad (1)$$

4 DELAY FUNCTIONS

As previously introduced in section 3.4, a *delay* defines how a transition, labeled by an event, is fired. Thus, if a transition becomes fireable at time t, a delay d is calculated, according to a corresponding *delay function*, and the transition is actually fired at time $t + d$, if it stays fireable from t to $t + d$.

A delay can be deterministic or stochastic and may depend on the state of the model. In general, simulation tools offer a lot of delay functions. Users have to choose the best matching ones and fill out parameters. Nevertheless and also in general, only a few ones are widely used for safety or reliability analyses (e.g.: exponential, Weibull, etc.). Whereas others ones are more specific and generally difficult to use: either to understand, either to fill out parameters. Due to this fact, the stochastic simulator of the AltaRica 3.0 project implements a few built-in delay functions and a generic mechanism to define specific ones point wisely.

4.1 *Representation of a delay function*

A *delay function* should be seen essentially as the inverse of an (invertible) cumulative probability distribution. A cumulative probability distribution is a non-decreasing function from the set of positive real numbers $\mathbb{R}_+$, representing the time, to the real range $[0;1]$ representing the probability.

A such function associates with each time $t \in \mathbb{R}_+$ the probability that a given event E (associated to the delay function) occurs before this time t. Given a probability $p \in [0;1]$, it is therefore possible to calculate the earliest date $t \in \mathbb{R}_+$ such that the given event E has the probability p to occur before the time t. The limit as the time goes to infinity of the Cumulative Probability Distribution may be less than 1. In that case, the delay function returns $+\infty$ for probabilities greater than the limit.

Stochastic delays functions are deterministic in the sense that they just take an oracle as parameter which is in charge of giving the probabilities.

As previously said, the stochastic simulator of the AltaRica 3.0 project implements in general a few built-in delay functions. They are the most widely used in Reliability analyses. For others delay functions, a generic mechanism allows users to describe them by means of a set of points interpolated in a triangular way.

4.2 *Built-in delay functions*

A Dirac delay is a deterministic delay. It means that the event E occurs after a fixed delay d. Thus for a probability $p \in [0,1]$, done by the oracle, the value of the delay (a time instant) is done by:

$$delay(p,d) = d \tag{2}$$

Another basic type of cumulative probability distribution is the constant distribution which corresponds to the point wise probability in Fault Trees. It means that the event occurs at time 0 with a probability q. For a probability $p \in [0,1]$, the value of the delay is done by:

$$delay(p, constant(q)) = \begin{cases} 0, & \text{if } p \leq q. \\ +\infty, & \text{otherwise.} \end{cases} \tag{3}$$

The most widely used probability distribution is the exponential distribution that corresponds to a Markovian hypothesis according to a rate r. For a probability $p \in [0,1]$, the value of the delay is done by:

$$delay(p, exponential(r)) = -\frac{log(1-p)}{r} \tag{4}$$

Another frequently used cumulative distribution function is the Weibull distribution. It takes two parameters: a shape factor β and a scale factor α. For a probability $p \in [0,1]$, the value of the delay is done by:

$$delay(p, Weibull(\alpha,\beta)) = \alpha \cdot (-log(1-\beta))^{\frac{1}{\beta}} \tag{5}$$

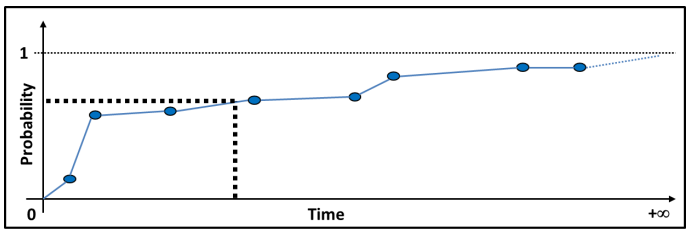

Figure 1: A Cumulative Probability Distribution.

4.3 *Generic mechanism for other delay functions*

A generic mechanism allows users to describe cumulative distribution function by means of a set of points interpolated in a triangular way. It considers a such set of points $D = \{(t_0,p_0);(t_1,p_1);\cdots;(t_n,p_n)\}$ in $\mathbb{R}_+ \times [0,1]$.

For a probability $p \in [0,1]$, the associated delay $delay(p,D) \in \mathbb{R}_+$, according to the set D of points, is done by the "place" of p between p_0 and p_n. It means that if it exists $i \in [0;n[$ such that $p_i \leq p \leq p_{i+1}$, thus:

$$delay(p,D) = \frac{(t_{i+1}-t_i)}{(p_{i+1}-p_i)} \cdot (p-p_i) + t_i \tag{6}$$

otherwise:

$$delay(p,D) = +\infty \tag{7}$$

Figure 1 illustrates this generic mechanism. Blue ovals represent points and blue lines, between ovals, represent the interpolation.

5 OBSERVERS

Observers correspond to indicators to be observed within the model. They are quantities calculated during simulations in order to perform reliability and safety analyses of the system to study. Because they only provide information on quantities to observe, they cannot be used to describe the behavior of the system.

Within the stochastic simulator of the AltaRica 3.0 project, observers can be defined in a easy and generic way. Then during simulations, they are updated after each transition firing. Two types of observers are considered: observers on events and observers on formulas.

As stochastic simulations consists on make statistics on simulations, these statistics are thus done under observers. As example, the average numbers of times an event has been fired.

5.1 *Observers on events*

For each defined event, an observer is automatically associated. For each such event observer and for one simulation, the stochastic simulator records the following: the number of times the event has been fired, the number of times it has been fireable, the time spent in a fireable state (before being fired or not) and the first date it is fired.

5.2 *Observers on formulas*

It is generally necessary to observe specific quantities of the system (e.g.: the production of a unit). An observer on formula is specifically designed for that. The definition of a such observer on a formula can involve variables or parameters of the model and other observers, under the condition that this introduces no circular definitions.

For each such observer on a formula and for one simulation, the stochastic simulator records the following: the different values taken by the formula, the time it spends having a given value and the first date it takes a given value.

5.3 *Statistics on observers*

Within the stochastic simulator, these statistics are made after all simulations are performed. At the end of all simulations, the recorded data are mixed and statistics are made during a final step. These statistics are different according to the type of observers.

For an observer on one event, statistics are the following: the average number of times an event has been fired, the average number of times it has been fireable, the average time spent in a fireable state (before being fired or not) and the average first date it is fired.

For an observer on one formula, statistics are the following: the average number of times the observers takes a given value, the average time it spends having a given value and the average first date it takes a given value.

6 COMPILATION TECHNIQUES

Monte Carlo simulation is based on several pseudo random histories representing behaviors of the system. In the following, a such history, representing a behavior of the system, is called a *simulation*. Simulations are generated histories representing behaviors of the system. So, to stabilize the measures (the estimates done by observers), it is necessary to obtain an important number of simulations, of sufficient length. It is particularly true when one wants to consider rare events.

Thus the occurring problem is the computation time to obtain all necessary simulations. This is the only limit of Monte Carlo simulation. But with nowadays computers, it is relatively easy to perform up to several millions of simulations, of reasonable length. Beyond, reduce the computation time of simulations gets an issue.

6.1 *Solutions to reduce computation time of simulations*

In order to make a simulation statistically efficient, and thus enhance efficiency of a stochastic simulator, techniques are classically based on the variance reduction (Rubinstein & Melamed 1998). The principle is to increase the precision of the estimates (i.e. to obtain a greater precision and smaller confidence intervals) that can be obtained for a given number of iterations. Nevertheless, methodologies based on these techniques depend on the structure of the system to study (Baca 1993, Cancela and Khadiri 1998).

Our solution is based on another technique. We use compilation techniques, as shown in (Khuu 2008). Generally, the simulator and the model to simulate (describing the system to study) are separated and the model is interpreted during simulations. By using simulation techniques, they are gathered at the (considered) programming language level. The principle, of such techniques, is to translate the model onto instructions of a programming language (e.g.: C or C++). These generated instructions represent parameters of a simulation kernel. In this way, a stochastic simulator, of the system to study, is produced by combining the generated instructions with the simulation kernel. Finally, optimization can be realized on the data structure of the model.

6.2 *Compilation techniques*

The stochastic simulator is implemented in C++. As illustrated in Figure 2, it is produced on two steps: a compilation of the model and a compilation of the simulator.

The compilation of the model is performed by the *model compiler*. This model compiler translates a GTS model in C++ classes, representing a set of instructions for the simulation kernel. During this step, first optimization can be realized. Reordering instructions within a block, or divide a block into sub-blocks. A such division can be performed for the assertion (called by all transitions) when specific ones do not evaluate the overall instructions.

Generated C++ classes are then gathered with the simulation kernel. This simulation kernel is also implemented in C++ as a static library. Finally, the overall is compiled to produce the stochastic simulator of the system to study. During this step, it is important to remark that the model and the simulation kernel are on the form of C++ source code. The power of the language is thus used to make optimization.

For the first version, we have consider two types of optimization, presented in the two next sub-parts. Of course, they do not represent the overall process of generation form a GTS model to C++ classes.

6.2.1 *Data structure of variables*

Assignments of values to variables can be performed efficiently by using appropriate data structures. The idea is to create, for each data type of variables (i.e.: boolean, integer, real and symbol), a table with size corresponding to the number of variables of the considered type. For example, if the model has two

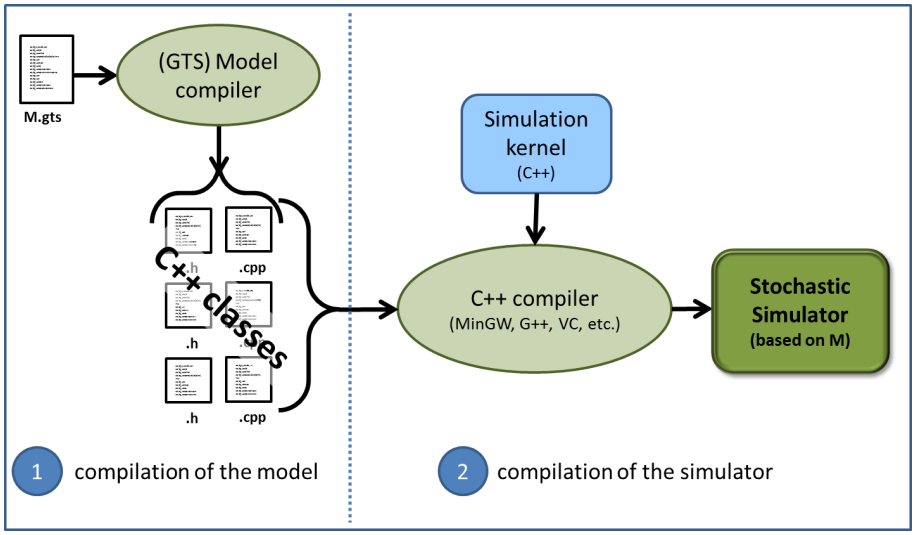

Figure 2: Structure of the stochastic simulator.

boolean variables `state1` and `state2` and one symbol variable `level`, two tables are created:

```
bool tabBoolVar[2];
char * tabSymbVar[1];
```

Then to fill tables, we use the preprocessor directive `#define`. For example for the previous boolean and symbol variables:

```
#define state1 0
#define state2 1
#define level 0
```

Finally to obtain the value of a variable, we just have to take it from the table with its name. For example with the variable state2:

```
tabBoolVar[state2]
```

which is replaced by the preprocessor, during the compilation step, by the instruction:

```
tabBoolVar[1]
```

6.2.2 *Expression evaluation*

Expressions are built from the set of variables and parameters (representing constants) with classical arithmetic, comparison (and some others) operators. For example, a such expression with variables of the previous part 6.2.1:

```
state1 and level == FULL
```

Evaluation of this expression is translated in the following C++ source code:

```
tabBoolVar[state1] && tabSymbVar[level]=="FULL"
```

This makes the most powerful optimization. In fact with classical simulation (i.e when the model is separated) there is an interpretation of expressions. Data structure of expressions is done according to the types (atomic, or composed) and the interpretation is performed according to the type of expression. Such construction must have to make references to several information and access to them is time consuming.

6.3 *Gains*

Comparisons were realized (Khuu 2008) between the stochastic simulator of AltaRica Data-Flow, where models are interpreted, and stochastic simulators generated from models. The programming languages used was C. Even if we consider the C++ programming language for our implementation, we present these results in order to emphasize the fact that compilation techniques drastically reduce the computation time of simulations. Comparisons were realized according to the kind of components of the system and also the kind of connections between them.

For reparable components, were failure and repair events follow the exponential function delay (with specific rates), the gain is about 35 when components are linked in series, and 30 when they are linked in parallel. These gains are stable from a number of 16 linked components. When the system to study combines both links, the gain is not stable. In the worst case (the minimal gain to obtain), it is about 44. These “huge” gains are explained because of the cost of flow propagation between components.

Systems with cold redundancies and finally two specific benchmarks are also studied.

- The Kawauchi and Rausand benchmark (Kawauchi and Rausand 2002) represents a production system with 8 assessment units of gas from an oil well. For both simulators, 10 000 simulations were performed, with a mission time to 17 520 hours (two years). Running time, to perform all simulations, for the interpreted simulator is 19.87 seconds; whereas for the compiled simulator, it is 1.11 seconds.

- An oil extraction installation (from Boiteau et al. (2006)) is also considered. The goal is to study its average production throughout a period of time. Also for both simulators, 10 000 simulations were performed, with a mission time to 8 760 hours (one year). Running time, to per-

form all simulations, is 388 seconds for the interpreted simulator; whereas it is 7.81 seconds for the compiled simulator.

These results confirm the benefits of using compilation techniques. The gain obtained for computation time of simulations are substantial.

7 CONCLUSION

For safety and reliability analyses, the Model-Based approach is gradually wining the trust of engineers and is still an active domain of research. The high-level modeling language AltaRica has been created to model systems far from the specifications of the systems under study; in order to be easy to design and to maintain throughout the life cycle of systems. Within the AltaRica 3.0 project, an entirely new version of the language, AltaRica 3.0, is developed. Its underlying mathematical model, Guarded Transition Systems, makes it possible to design acausal components and to handle looped systems. The project includes the development of a complete set of freeware authoring and assessment tools is planed, so to make them available to a wide audience.

One of these tools, of the AltaRica 3.0 project, is the stochastic simulator. Its first version is now implemented and it was designed by taking into account original features. We have consider delay functions in a generic manner: a few built-in delay functions, representing the most widely used in safety and reliability analyses, are implemented by the simulator. For more specific ones, a generic mechanism allows users to define them point wisely. Observers, corresponding to indicators observed during simulations in order to perform reliability and safety analyses, are also defined in a generic way. Two types are considered: observers on events and observers on formulas. Compilation techniques are used to reduce computation time of simulations. We have shown the "huge" gains obtained by using these techniques.

First tests are actually achieved. We check simulation functions for different kind of models and realize tests on benchmarks. These results will be presented in forthcoming papers. For future works, we will focus on a specific on-the-fly calculation engine for observers. In fact and as explain before, actually the stochastic simulator provides results (of observers) in the form of raw data after each simulation. Statistics are made during a final step: when all simulations are performed. Thus specific statistical calculation on these data could, and probably should (because of the length), be achieved during simulations.

REFERENCES

Baca, A. (1993). Examples of monte carlo methods in reliability estimation based on reduction of prior information. *IEEE Transactions on Reliability 42*(4), 645–649.

Boiteau, M., Y. Dutuit, A. Rauzy, & J.-P. Signoret (2006). The altarica data-flow language in use: Assessment of production availability of a multistates system. *Reliability Engineering and System Safety 91*, 747–755.

Cancela, H. & M. E. Khadiri (1998). Series-parallel reductions in monte carlo network-reliability evaluation. *IEEE Transactions on Reliability 47*(2), 159–164.

Coates, R., G. Janacek, & K. Lever (1988). Monte carlo simulation and random number generation. In *IEEE Transaction on Selected Area in Communications*, Volume 6(1), pp. 313–320.

Dubi, A. (2000). *Monte Carlo Applications in Systems Engineering*. John Wiley and Sons Ltd.

Kawauchi, Y. & M. Rausand (2002). A new approach to production regularity assessment in the oil and chemical industries. *Reliability Engineering and System Safety 75*, 379–388.

Khuu, M. (2008). *Contribution à l'accélération de la simulation stochastique sur des modèles AltaRica Data Flow*. French phd thesis, Université de la Méditerranée (Aix-Marseille II).

Liu, J. (2008). *Monte Carlo Strategies in Scientific Computing*. Springer.

Prosvirnova, T. & A. Rauzy (2012, Octobre). Système de transitions gardées : formalisme pivot de modélisation pour la sûreté de fonctionnement. In J. Barbet (Ed.), *Actes du Congrès Lambda-Mu 18*.

Rauzy, A. (2008). Guarded transition systems: a new states/events formalism for reliability studies. *Journal of Risk and Reliability 222*(4), 495–505.

Rubinstein, R. & B. Melamed (1998). *Modern Simulation and Modeling*. John Wiley and Sons Ltd.