The AltaRica 3.0 project for Model-Based Safety Assessment

Michel Batteux, Tatiana Prosvirnova, Antoine Rauzy
LIX - Ecole Polytechnique
route de Saclay,
91128 Palaiseau cedex, FRANCE
Email: name@lix.polytechnique.fr

Leïla Kloul
PRISM - Université de Versailles St-Quentin-en-Yvelines
45 avenue des États-Unis, Bâtiment Descartes,
78035 Versailles Cedex, FRANCE
Email: leila.kloul@prism.uvsq.fr

***Abstract*—"Traditional" risk modeling formalisms (e.g. FMEA, Fault Trees, Markov Processes, etc.) are well mastered by safety analysts. Efficient algorithms and tools are available. However, models designed with these formalisms are far from the specifications of the systems under study. They are consequently hard to design and to maintain throughout the life cycle of systems. The high-level modeling language AltaRica has been created to tackle this problem.**

The objective of the AltaRica 3.0 project is to design a new version of AltaRica, and to develop a complete set of authoring, simulation and assessment tools to perform safety analyses: virtual experiments on systems, via models, calculation of different kinds of reliability indicators, etc. AltaRica 3.0 improves significantly the expressive power of AltaRica Data-Flow without decreasing the efficiency of its assessment algorithms. Prototypes of a Fault Tree compiler, a stochastic and a stepwise simulators have been already developed. Other tools are under specification or implementation.

I. Introduction

The Model-Based approach for Safety and Reliability Analysis is gradually wining the trust of engineers but 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 to assess these models. 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, they are hard to design and to maintain throughout the life cycle of systems. A small change in the specifications may require revisiting completely the safety models, which is both resource consuming and error prone.

The high-level modeling language AltaRica Data-Flow ([1], [2]) has been created to tackle this problem. AltaRica Data-Flow models are made of hierarchies of reusable components. Graphical representations are associated with components, making models visually very close to Process and Instrumentation Diagrams. AltaRica Data-Flow is at the core of several Integrated Modeling and Simulation Environments: Cecilia OCAS (Dassault Aviation), Simfia (EADS Apsys), and Safety Designer (Dassault Systèmes). Several successful industrial experiments using AltaRica Data-Flow have been reported ([3], [4], [5], [6]). For example, AltaRica Data-Flow was used to certify the flight control system of the aircraft Falcon 7X (Dassault Aviation). In a word, AltaRica Data-Flow has reached an industrial maturity.

However, more than ten years of experience showed that both the language and the assessment tools can be improved. AltaRica 3.0 is an entirely new version of the language. Its underlying mathematical model – Guarded Transition Systems ([7], [8]) – makes it possible to design acausal components and to handle looped systems. A complete set of freeware authoring and assessment tools is under development, so to make them available to a wide audience.

The aim of this article is to present the AltaRica 3.0 project. It is organized as follows.

Section II presents related works in order to position the project. Section III introduces the AltaRica 3.0 project. Section IV presents the new version of AltaRica modeling language, AltaRica 3.0, and Section V introduces the underlying formalism - Guarded Transition Systems (GTS). In Section VI, an example of AltaRica 3.0 model and the associated GTS is given. In Section VII, some assessment tools are presented. Finally in the last part, a conclusion summarizes this AltaRica 3.0 project and outlines interesting directions for future works.

II. Related Works

Two approaches for (high-level) Model-Based Safety Assessment can be found in literature. The first one consists in creating extensions of high-level modeling languages used in other domains. The second approach consists in defining domain specific languages, dedicated to Safety Analyses.

In the first category, we can find [9] which adds an Error Model annex to the modeling formalism for embedded real-time systems AADL. In the same way the HiP-HOPS workbench [10] enables to add reliability data to models imported from different modeling tools: Matlab/SIMULINK, Eclipse-based UML tools, etc., and then to automatically generate Fault Trees and FMEA tables.

Similarly, translations have been defined from specialized UML/SysML models to Fault Trees or Petri nets [11]. In [12], the functional design phase, using SysML, is combined with commonly used reliability techniques (i.e. FMEA and construction of AltaRica Data-Flow models).

In the second category, we can find Figaro [13], developed by EDF R&D. It is a textual modeling language dedicated

to dependability assessment of complex systems. It combines object-orientation languages features, such as inheritance, and first order production rules (interaction and occurrence rules). It is used as a description language to create knowledge bases for the workbench KB3 [14], to automatically perform systems dependability assessment: Monte-Carlo simulation, Markov Chain generation, quantification and generation of critical sequences, etc.

In between the two categories, we can find SAML (Safety Analysis Modeling Language) [15], which is a synchronous language. It expresses a model in terms of finite stochastic state automata and its semantics is defined as Markov decision process. S3E is a design and verification environment focused on SAML models. It provides a stochastic simulator and translators to the input languages of the model-checkers PRISM and NuSMV.

All of these proposals are interesting, however AltaRica provides specific constructs and assessment tools that we shall describe now.

III. Overview of the AltaRica 3.0 project

The objective of the AltaRica 3.0 project is to provide a set of authoring, simulation and assessment tools to perform Safety Analyses. Figure 1 presents the overview of the project.

The new version of AltaRica modeling language is in the heart of this project. It supports modeling of looped systems and bidirectional flows. This new version significantly increases the expressive power of the previous one without decreasing the efficiency of its assessment algorithms.

Many high-level languages and tools for systems modeling and simulation have been proposed: SysML/UML for system architecture modeling, Modelica [16] or Matlab Simulink for system dynamic behavior modeling, Lustre [17] for synchronous systems modeling, AADL [18] for embedded real-time systems modeling, etc. All of them are efficiently used in their respective domains of application but do not really correspond to the needs of Safety and Reliability engineers. The attempts to extend these existing modeling languages for Safety Analyses, such as, for example, those cited in Section II, are interesting, but the idea of having a universal language and a unique model encompassing all of the properties of the system is not realistic and, from our point of view, is an engineering dead-end. AltaRica 3.0 seeks to meet the needs of Safety and Reliability engineers and proposes a good trade-off between the expressive power and the efficiency of the assessment algorithms.

AltaRica 3.0 models are compiled into a low-level formalism: Guarded Transition Systems. Guarded Transition Systems is a states/transitions formalism that generalizes classical safety formalisms, such as Reliability Block Diagrams, Petri Nets and Markov chains. It is a pivot formalism for Safety Analyses: other safety models, not only AltaRica 3.0, can be compiled into Guarded Transition Systems to take benefits from assessment tools.

The assessment tools for Guarded Transition Systems include a Fault Tree compiler to perform Fault Tree Analysis (FTA), a Markov chain generator, a stochastic and a stepwise simulators, a model-checker, and a reliability allocation module. Prototypes of three of them have been already developed. They will be described later in Section VII. Other tools are under specification or implementation.

These tools will be distributed under a free license. They 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).

In the context of certification process and safety critical systems the ability to perform cross check verification of the obtained results is of great interest. Also, Safety Analyses are complex (in the sens of computational complexity theory). Even the calculation of the top event probability is #P-hard. Different assessment tools for Safety Analyses enable to push the limits of this complexity.

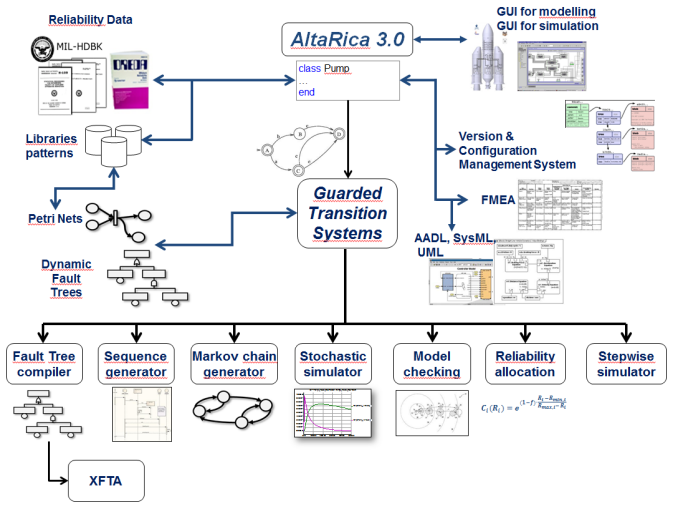

Fig. 1. Overview of the AltaRica 3.0 project

IV. AltaRica 3.0 Modeling Language

The previous version of AltaRica modeling language (i.e. 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 complex systems.

Thus, a new version of AltaRica, so-called AltaRica 3.0, is currently under specification. It improves AltaRica Data-Flow into two directions:

1) Its semantic is based on the new underlying mathematical model: Guarded Transition Systems.
2) 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 (i.e. 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).

AltaRica 3.0 is a prototype oriented modeling language (see e.g. [19] for a discussion on objects versus prototypes). We believe that prototype orientation is the paradigm that corresponds the best with cognitive processes of engineers. Prototype orientation makes it possible to separate the knowledge into two distinct spaces: the stabilized knowledge which is incorporated into libraries of on-the-shelf modeling components; and the sandbox in which the system under study is modeled. In the sandbox, many components are unique; some others are instances of reusable components. With prototype-orientation, models can be reused in two ways: at component level by instantiating on-the-shelf components and at system level by cloning and modifying a model designed for a previous project.

V. Pivot formalism for Safety Analyses: Guarded Transition Systems

Before any assessment, AltaRica models are "flattened" (i.e. reduced to a single class without sub-classes). This "flattened" class produces a GTS model. First introduced in [7], GTS 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 ([8]) makes it possible to represent components with bidirectional flows.

A. Definition

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

- $V = S \uplus F$ is a set of variables, divided into 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.
- 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, denoted $e : G \rightarrow P$, 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 $e : G \rightarrow P$ 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.

B. Instructions

Both assertions and actions of transitions are described by means of instructions. There are basically four types of instructions:

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

State variables can occur as the left member of an assignment only in the action of a transition. Flow variables can occur as the left member of an assignment 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 $e : G \rightarrow P$. Applying the instruction P to the variable assignment σ consists in calculating a new variable assignment τ. The 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 $I_1, \ldots, I_n$ 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.

C. Reachability graph

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. This graph is constructed in the following way.

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 $Fire(e : G \rightarrow P, A, \sigma)$ defined as follows:

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

The so-called reachability graph $\Gamma = (\Sigma, \Theta)$ is the smallest Kripke structure, such that the following is verified:

1) $\sigma_0 = Propagate(A, \iota, \iota) \in \Sigma$. σ_0 is the initial state of the Kripke structure.
2) If $\sigma \in \Sigma$ and $\exists t = \langle e, G, P \rangle \in T$, such that the guard G is verified in σ then the state $\tau = Fire(P, A, \iota, \sigma) \in \Sigma$ and the transition $(\sigma, e, \tau) \in \Theta$,

The calculation of $\Gamma = (\Sigma, \Theta)$ may raise errors. A well designed Guarded Transition Systems avoids this problem.

D. 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 a delay and a weight, called expectation.

A delay 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 $t + d$.

An expectation is used to determine the probability that the transition is fired in case of several transitions are fireable at the same date.

As an illustration, in the example given in Section VI, the transition $failure$ is a timed stochastic transition that obeys to the exponential distribution with a failure rate $lambda$.

VI. Example

As example, we consider a sub-system composed of two valves connected together in series. Figure 2 shows this example. *rightFlow* and *leftFlow* express flows of the component (we will see later that each valve contains its own flows from left and from right). They are bidirectional flows, so we do not name them with words "input" or "output".

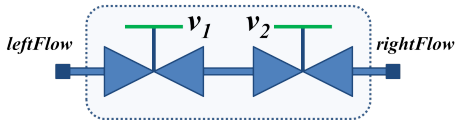

Fig. 2. The component *TwoValves*.

A. AltaRica 3.0 model of a valve

A valve is represented by a class. It will be used as a piece to build the component *TwoValves*. We consider a valve that can be opened or closed (it corresponds to the normal behavior). It can also be blocked (it is the failure behavior). The AltaRica 3.0 model of a valve is the following:

```
class Valve
  Boolean closed (init = true), blocked (init = false);
  Real rightFlow (reset = 0), leftFlow (reset = 0);
  parameter Real lambda = 0.001;
  event open, close;
  event failure (delay = exponential(lambda));
  transition
    open: closed and not blocked -> closed := false;
    close: not closed and not blocked -> closed := true;
    failure: not blocked -> blocked := true;
  assertion
    if not closed then leftFlow :=: rightFlow;
end
```

A class *Valve* contains two state variables *closed* and *blocked*; and two flow variables *rightFlow* and *leftFlow*. Events *open*, *close* and *failure* are declared and then used to define transitions. The event *failure* occurs according to an exponential delay with a parameter *lambda*. The behavior of a valve is then defined by transitions (to update its state) and the assertion (to propagate flows). The operator ":=:" expresses a bidirectional flow circulating through the valve.

B. AltaRica 3.0 model of the sub-system TwoValves

The component *TwoValves* contains two instances of the class *Valve*. It can be failed because of failures of both valves v_1 or v_2. The AltaRica 3.0 model of this component *TwoValves* is the following:

```
class TwoValves
  Valve v1, v2;
  Real rightFlow (reset = 0), leftFlow (reset = 0);
  event ccf;
  transition
    ccf: ?v1.failure & ?v2.failure;
  assertion
    leftFlow :=: v1.leftFlow;
    rightFlow :=: v2.rightFlow;
    v1.rightFlow :=: v2.leftFlow;
end
```

Operator "&" expresses synchronizations between events. The event *ccf* represents the common cause failure of the two valves. It is expressed by means of the synchronization of the events *v1.failure* and *v2.failure*. The event *ccf* may occur when at least one of two valves is working. Operator "?" means that the event is optional for the synchronization.

C. The GTS model

The following "flattened" class encodes the GTS of the component *TwoValves*:

```
class TwoValves
  variable
    Boolean v1.closed (init = true);
    Boolean v2.closed (init = true);
    Boolean v1.blocked (init = false);
    Boolean v2.blocked (init = false);
    Real v1.rightFlow (reset = 0);
    Real v2.rightFlow (reset = 0);
    Real v1.leftFlow (reset = 0);
    Real v2.leftFlow (reset = 0);
    Real rightFlow (reset = 0);
    Real leftFlow (reset = 0);
  parameter
    Real v1.lambda = 0.001;
    Real v2.lambda = 0.001;
  event
    v1.open;
    v2.open;
    v1.close;
    v2.close;
    v1.failure (delay = exponential(v1.lambda));
    v2.failure (delay = exponential(v2.lambda));
    ccf;
  transition
```

```
    v1.open: v1.closed -> v1.closed := false;
    v1.close: not v1.closed -> v1.closed := true;
    v2.open: v2.closed -> v2.closed := false;
    v2.close: not v2.closed -> v2.closed := true;
    ccf: not v1.blocked or not v2.blocked ->
        {
          if not v1.blocked then v1.blocked := true;
          if not v2.blocked then v2.blocked := true;
        }
    v1.failure: not v1.blocked -> v1.blocked := true;
    v2.failure: not v2.blocked -> v2.blocked := true;
  assertion
    leftFlow := v1.leftFlow;
    v1.leftFlow := leftFlow;
    rightFlow := v2.rightFlow;
    v2.rightFlow := rightFlow;
    v1.rightFlow := v2.leftFlow;
    v2.leftFlow := v1.rightFlow;
end
```

Note that this GTS model is more bigger than the previous AltaRica 3.0 model. Nevertheless, users never use this formalism to describe a system. A specific tool is devoted to compile AltaRica 3.0 models to GTS.

VII. Assessment tools

The AltaRica 3.0 project includes a set of assessment tools. All these tools take, as input, a GTS model. Thus, the user first designs an AltaRica 3.0 model. Then, this model is compiled into its flattened version: the corresponding GTS model.

As shown by the previous figure 1, the set of assessment tools includes a Fault Tree compiler to perform Fault Tree Analysis, a Markov chain generator, a stochastic and a stepwise simulators, a sequence generator, a model-checker and a reliability allocation module. Prototypes of a compiler to Fault Trees, of a stepwise and a stochastic simulators have been already developed. In the sequel, we will present them.

A. Compiler to Fault Trees

Fault trees are widely used to perform Safety Analyses and some regulation authorities require to use them to support the certification process. From a GTS model, it is possible to generate corresponding Fault Trees (FT), i.e. to transform a states/transitions model into a set of Boolean formulae. It may seem inefficient at a first glance to use a states/transitions formalism to end up with a Fault Tree. However in practice, it is of great interest. It is easier and less time consuming to automatically generate Fault Trees from high-level models rather than create them from scratch. High-level models improves greatly the design, the sharing and the maintenance of models. The algorithm of compilation to Fault Trees for AltaRica Data-Flow, described in [1], can be extended to a general case. As it is illustrated Figure 3, it includes 3 steps:

1) The GTS model is partitioned into independent GTS and an independent assertion.
2) Reachability graphs of each independent GTS are calculated.
3) Reachability graphs and the assertion are separately compiled into Boolean equations.

Partitioning is a key point of the algorithm that ensures its efficiency. In practice, components of a system fail in general in a relatively independent way. In that case the partitioning is possible. If the GTS is combinatorial, its compilation to Fault Trees is efficient and does not loose information.

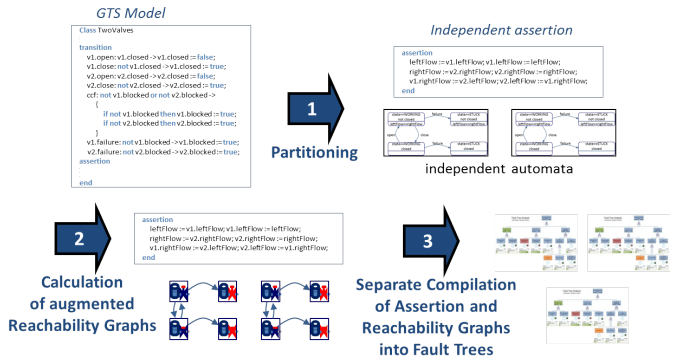

Fig. 3. Algorithm of compilation of GTS to FT

The generated Fault Tree can be assessed with any Fault Tree calculation engine supporting Open-PSA format [20]. For example, it will be possible to use XFTA [21] to calculate minimal cutsets, events probabilities, importance factors for the generated Fault Tree.

B. Stepwise simulator

The stepwise simulator enables to perform an interactive step by step simulation of a GTS model. This interactive tool can be very useful to debug models, to play different failure scenarios, etc. The stepwise simulator can be coupled with a graphical simulator as illustrated in [22]. The graphical simulation of models can be used to perform virtual experiments on systems via models, helping to better understand the system behavior.

The first version of the interactive stepwise simulator supports commands:

- to display the information about the simulated model (variables and their values, observers and their values, transitions fireable in the current state, history of fired transitions since the beginning of the simulation, etc.);
- to perform action on the simulated model (fire a transition, cancel the last fired transition, restart the simulation from the initial state, etc.).

C. Stochastic simulator

Stochastic simulation is a basic tool for safety analyses of systems. It provides fine results to calculate reliability indicators, even with complex systems. The principle is to run many pseudo-random histories of the behavior of the system and to make statistics on these histories. The stochastic simulator of the AltaRica 3.0 project has been designed by taking into account original features.

We use compilation techniques to reduce computation time of simulations. In fact, the only limit of stochastic simulation is the number of histories, and their length, necessary to stabilize the measures. But with nowadays computers, it is relatively easy to perform up to several millions of histories (of reasonable length). Beyond, the computation time gets an issue and with that respect, compilation technique may be of a great help. Thus the considered GTS model is translated into

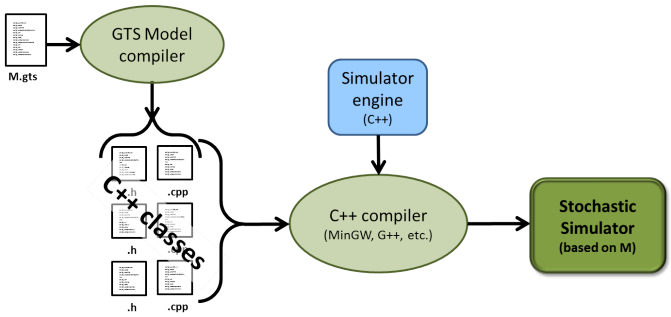

Fig. 4. Overview of the stochastic simulator

C++ classes, representing a set of instructions for the simulator engine. Then, combined with this simulator engine, they are compiled to constitute the stochastic simulator of the system to study. Figure 4 illustrates this purpose.

We have implemented a generic mechanism to allow users to define specific delays. In general, simulation tools offer a lot of stochastic delays. Users have to choose the best matching one and fill out parameters. A few ones are widely used for safety/reliability analyses (e.g. exponential, Weibull, etc.), but others are generally difficult to use: difficult to understand or to fill out parameters. Due to this fact, the stochastic simulator implements the widely used delay functions (previously mentioned). For other specific ones, a generic mechanism allows to describe them by means of a set of points interpolated in a triangular way. For a set of points $Del = \{(t_0, p_0); (t_1, p_1); \cdots; (t_n, p_n)\}$, in $\mathbb{R}_+ \times [0, 1]$ (Del meaning "delay") and a probability $p \in [0, 1]$, the associated delay $d_{Del} \in \mathbb{R}_+$ is done by:

$$\frac{(t_{i+1} - t_i)}{(p_{i+1} - p_i)} \cdot (p - p_i) + t_i$$

if it exists $i \in [0; n[\setminus p_i \leq p \leq p_{i+1}$. Otherwise it is done by $+\infty$.

VIII. Conclusion

In this article, we have presented the AltaRica 3.0 project. The aim of this project is to develop a complete set of tools (distributed under a free license) to create, edit, check, simulate, debug and assess AltaRica 3.0 models, thus making them available to a large (industrial and academic) community.

AltaRica 3.0 modeling language increases the expressive power of the previous one without decreasing the efficiency of assessment algorithms. It makes it possible to model looped systems and components with bidirectional flows. First versions of assessment tools include: a Fault Tree compiler, stepwise and stochastic simulators.

Forthcoming works will focus, amongst others, on the implementation of other assessment tools and on the redaction of pedagogical materials, including benchmark tests.

References

[1] A. Rauzy, "Modes automata and their compilation into fault trees," *Reliability Engineering and System Safety*, vol. 78, pp. 1–12, 2002.

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

[3] R. Bernard, J.-J. Aubert, P. Bieber, C. Merlini, and S. Metge, "Experiments in model-based safety analysis: flight controls," in *Proceedings of IFAC workshop on Dependable Control of Discrete Systems, Cachan*, 2007.

[4] P. Bieber, J.-P. Blanquart, G. Durrieu, D. Lesens, J. Lucotte, F. Tardy, M. Turin, C. Seguin, and E. Conquet, "Integration of formal fault analysis in assert: Case studies and lessons learnt," in *Proceedings of 4th European Congress Embedded Real Time Software, ERTS 2008*, Toulouse (France), January 2008.

[5] X. Quayzin and E. Arbaretier, "Performance modeling of a surveillance mission," in *European Safety and Reliability Conference, ESREL 2009*, Praha (Czech Republic), September 2009.

[6] R. Bernard, S. Metge, F. Pouzolz, P. Bieber, A. Griffault, and M. Zeitoun, "Altarica refinement for heterogeneous granularity model analysis," in *Actes du congrès Lambda-Mu'16*, Avignon (France), October 2008.

[7] A. Rauzy, "Guarded transition systems: a new states/events formalism for reliability studies," *Journal of Risk and Reliability*, vol. 222, no. 4, pp. 495–505, 2008.

[8] T. Prosvirnova and A. Rauzy, "Guarded transition systems: Pivot modelling formalism for safety analysis," in *Actes du Congrès Lambda-Mu 18*, J. Barbet, Ed., Octobre 2012.

[9] P. Feiler and A. Rugina, "Dependability modeling with the architecture analysis & design language (aadl)," Carnegie Mellon University, Tech. Rep., 2007.

[10] A. Pasquini, Y. Papadopoulos, and J. McDermid, "Hierarchically performed hazard origin and propagation studies," *Computer Safety, Reliability and Security*, vol. 1698 of LNCS, pp. 688–688, 1999.

[11] S. Bernardi, S. Donatelli, and J. Merseguer, "From uml sequence diagrams and statecharts to analyzable petri net models," in *In Proceedings of the Third International Workshop on Software on Performance*, 2002.

[12] P. David, V. Idasiak, and F. Kratz, "Reliability study of complex physical systems using sysml," pp. 431–450, 2010.

[13] M. Bouissou, H. Bouhadana, M. Bannelier, and N. Villatte, "Knowledge modelling and reliability processing: presentation of the figaro modelling language and associated tools," in *Proceedings of Safecomp'91*, 1991.

[14] M. Bouissou, "Automated dependability analysis of complex systems with the kb3 workbench: the experience of edf r&d," in *Proceedings of the International Conference on Energy and Environment*, 2005.

[15] M. Güdemann and F. Ortmeier, "A framework for qualitative and quantitative model-based safety analysis," in *Proceedings of 12th High Assurance System Engineering Symposium*, 2010, p. 132âĂŞ141.

[16] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. John Wiley & Sons Inc, 2004.

[17] P. R. N. Halbwachs, P. Caspi and D. Pilaud, "The synchronous dataflow programming language lustre," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, 1991.

[18] P. Feiler, D. Gluch, and J. Hudak, "The architecture analysis & design language (aadl): An introduction," Carnegie Mellon University, Tech. Rep., 2006.

[19] J. Noble, A. Taivalsaari, and I. Moore, *Prototype-Based Programming: Concepts, Languages and Applications*. Springer-Verlag, 1999.

[20] M. Hibti, T. Friedlhuber, and A. Rauzy, "Overview of the open psa platform," in *Proceedings of International Joint Conference PSAM'11/ESREL'12*, R. Virolainen, Ed., June 2012.

[21] A. Rauzy, "Anatomy of an efficient fault tree assessment engine," in *Proceedings of International Joint Conference PSAM'11/ESREL'12*, R. Virolainen, Ed., June 2012.

[22] B. Perrot, T. Prosvirnova, A. Rauzy, J.-P. S. d'Izarn, and R. Schoening, "Expériences de couplages de modèles AltaRica avec des interfaces métiers," in *Actes du congrès LambdaMu'17 (actes électroniques)*, E. Fadier, Ed. IMdR, October 2010.