Génération automatique d’équations Booléennes stochastiques à partir de modèles AltaRica 3.0 : vers une meilleure lisibilité des modèles générés

Automatic generation of stochastic Boolean equations from AltaRica 3.0 models: towards a better readability of the generated models

BATTEUX Michel
Systemic Intelligence
Paris, France
michel.batteux@systemic-intelligence.net

PROSVIRNOVA Tatiana
ONERA/DTIS, Université de Toulouse
Toulouse, France
tatiana.prosvirnova@onera.fr

RAUZY Antoine
NTNU-MTP
Trondheim, Norvège
antoine.rauzy@ntnu.no

Résumé — AltaRica 3.0 est un langage de modélisation dédié aux analyses probabilistes de sécurité de systèmes techniques complexes. L’équation « S2ML + GTS = AltaRica 3.0 » est une bonne façon de le présenter. AltaRica 3.0 résulte en effet de la combinaison de S2ML (System Structure Modelling Language), un ensemble de primitives orientées objet et orientées prototype permettant de structurer les modèles avec le cadre mathématique des GTS (Guarded Transition Systems). L’atelier de modélisation AltaRica 3.0 fournit plusieurs outils de traitement de modèles AltaRica 3.0 : un simulateur interactif, un simulateur stochastique, un générateur de séquence critiques ainsi qu’un compilateur vers les systèmes d’équations Booléennes stochastiques, le cadre mathématique sous-jacent aux arbres de défaillance et aux blocs diagrammes de fiabilité. L’objectif de cette communication est de présenter les améliorations que nous avons récemment apportées à ce dernier outil.

Mots-clefs — MBSA, AltaRica 3.0, génération automatique, équations Booléennes stochastiques, Blocs diagrammes de fiabilité, Arbres de défaillance

Abstract — AltaRica 3.0 is a modelling language dedicated to probabilistic safety analyses of complex technical systems. AltaRica 3.0 is the result of the combination of S2ML (System Structure Modelling Language), a set of object-oriented and prototype-oriented constructs to structure models, and the mathematical framework of GTS (Guarded Transition Systems). AltaRica 3.0 Workshop is an integrated modelling environment that provides several tools for processing AltaRica 3.0 models: an interactive simulator, a stochastic simulator, a generator of critical sequences as well as a compiler to stochastic Boolean equations, the underlying mathematical framework of fault trees and reliability block diagrams. The goal of this communication is to present the improvements that we have recently made to this last tool.

Keywords — MBSA, AltaRica 3.0, automatic generation, stochastic Boolean equations, Reliability Block Diagrams, Fault Trees.

I. INTRODUCTION

AltaRica 3.0 is a modelling language dedicated to probabilistic safety analyses of complex technical systems [1]. AltaRica 3.0 is the result of the combination of S2ML (System Structure Modelling Language), a set of object-oriented and prototype-oriented constructs to structure models [2], and the mathematical framework of GTS (Guarded Transition Systems) ([4],[5]). AltaRica 3.0 Workshop is an integrated modelling environment that provides several tools for processing AltaRica 3.0 models: an interactive simulator, a stochastic simulator, a generator of critical sequences as well as a compiler to stochastic Boolean equations, the underlying mathematical framework of fault trees and reliability block diagrams. The goal of this communication is to present the improvements that we have recently made to this last tool.

Automatic generation of stochastic Boolean equations from high-level models has many advantages. First, the same AltaRica model can be used to study several safety objectives. Second, the high-level model better reflects the architecture of the studied system and therefore is easier to develop and to maintain than the Boolean models.

The general principle of the compilation of AltaRica models towards systems of stochastic Boolean equations was stated in 2002 for AltaRica Data-Flow [7] and extended to AltaRica 3.0 in 2015 [8]. The compilation algorithm has several steps. The first step, called "flattening", consists in compiling the AltaRica model into a single system of guarded transitions. This first step loses the initial structure of the model that reflects the architecture of the studied system. The generated system of stochastic Boolean equations gives the expected results: the minimal cuts extracted from this system are those that the analyst expects. However, the generated Boolean model is very difficult to read by the analyst and very far from the one that the latter could have written "by hand".

We recently made several improvements to this compilation algorithm. We added a new step to the compilation process. This step, which could be called "inflating", produces the opposite effect of "flattening": it reinjects the generated stochastic Boolean equations into the original structure of the model. This makes it possible to obtain a Boolean model close to a hierarchical reliability block diagram, which is directly readable by the analyst.

This new compilation algorithm reinforces the AltaRica 3.0 technology and thereby the so-called model-based approach for dependability analyses.

The reminder of this article is organized as follows. Section II presents an example, which is used to illustrate the compilation algorithm. Section III introduces S2ML + X family of modelling languages. Section IV describes the compilation algorithm and its improvement. Section V concludes this article.

II. ILLUSTRATIVE EXAMPLE : A TRACKING SYSTEM

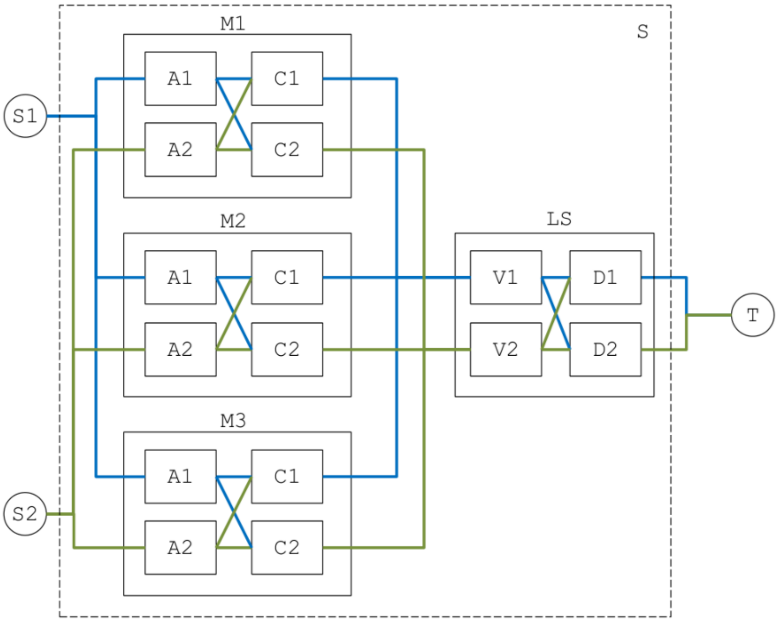

Fig. 1. A Tracking system

Consider the tracking system pictured in Fig. 1. The system is taken from [6]. This highly redundant system processes information coming from two redundant sources S1and S2 (external to the system). The information coming from each source is acquired in triplicated acquisition modules M1, M2 and M3. Each acquisition block consists of two acquisition chains, one for each source. Each chain consists itself of an acquisition block Ai and a calculator Ci. Results of calculations are sent to two voters V1 and V2 working according to a 2-out-of-3 logic.

Finally, the outputs of the two voters are aggregated into two calculators D1 and D2 that send the information to the target T (external to the system). Voters V1 and V2 and calculators D1 and D2 are part of the same logic solver LS.

We assume that all the components may fail in operation with failure rates given in TABLE I. Failures of components external to the system (S1, S2 and T) are not considered in this study.

The failure condition of interest occurs when the target T does not receive any information from the sources S1 and S2.

In this article, we first show how this case study can be easily represented with AltaRica 3.0 modelling language. Second, we illustrate how it is compiled into systems of stochastic Boolean equations.

TABLE I. FAILURE DATA

Component	**Failure data**	
	Probability distribution	***Parameters***
Acquisition unit	Exponential	λ=1.23e-4
Calculator	Weibull	α=5.67e+4, β=3
Voter	Exponential	λ=2.64e-7

III. S2ML + X FAMILY OF MODELLING LANGUAGES

Modelling languages used in different engineering domains (e.g. Modelica, Lustre, AltaRica, etc.) are made of two parts:

- A set of constructs to structure models, i.e. to organize models in order to make them easily readable, maintainable and reusable;
- An underlying mathematical framework describing the behaviour (e.g. linear differential equations, guarded transition systems, stochastic Boolean equations, etc.).

In the formula S2ML + X [3], S2ML stands for System Structure modelling language, a set of constructs to structure models and X stands for any mathematical framework describing system behaviour.
In safety analyses, modelling formalism can be divided in two categories:

- Combinatorial (also called Boolean or static), and
- State/transition formalisms (also called dynamic).

Examples of combinatorial formalisms are well known Fault Trees and Reliability block diagrams. Examples of state/transition formalisms are Markov chains, stochastic Petri nets and AltaRica. In the following, we present two modelling languages: AltaRica 3.0 [1] and new Open-PSA [9]. Both use S2ML for their structural part. For the behavioural part AltaRica 3.0 is based on Guarded Transition Systems and belongs to State/Transition category, and new Open-PSA is based on Stochastic Boolean Equations (SBE) and belongs to combinatorial category.

A. S2ML (System Structure Modelling language)

S2ML is a modelling language that provides a set of constructs to structure models, i.e. to organize models in order to make them easily readable, maintainable and reusable [2].

S2ML provides four basic elements:

- Ports, basic objects of models used to represent variables, events, parameters, equations and so on;
- Connections, used to describe relations existing between ports;
- Blocks, containers for ports, connections, blocks and other elements;
- Attributes, couples of name and value, used to associate information to ports, connections and blocks.

The basic structural construct is a block, also called a prototype. A block is a container for variables, parameters and all the other modeling artifacts. The simplest structuring relation is the composition. A block may be composed of several other blocks. Classical safety analysis formalisms, such as Fault Trees and Reliability Block Diagrams, use only blocks and composition for structuring models.
In order to be able to reuse blocks, structured programming languages introduce the notions of class and instantiation of classes. A class is a reusable “on-the-shelf” block, which is stored in a library and can be reused everywhere in the model via instantiation.
In some cases, it is necessary to modify or to extend a modeling unit (a class or a block) without instantiation. It can be achieved via inheritance relation introduced in object-oriented programming languages. If a modeling unit A inherits from a modeling unit B, then A contains all the characteristics of B and adds some new characteristics.
There are cases where the same component is used in several places or to contribute to different functions of the system. In other words, a modeling unit is shared between several other modeling units. This kind of “uses” relation between modeling units is called aggregation.
In object- oriented programming languages, the reuse of modeling units is done by means of instantiation of classes. In modeling languages using only blocks (called prototype-oriented languages), the reuse of blocks is also possible. It is achieved via the notion of cloning. If a block A is a clone of a block B, then the block A has exactly the same characteristics as the block B.
To summarize, S2ML proposed the following constructs to organize and structure models:

- Two types of modeling units: block and class;
- Three structural relations: composition, inheritance and aggregation; and
- Two mechanisms making possible to reuse modeling elements: prototype/cloning and class/instantiation.

These constructs originate from programming languages.

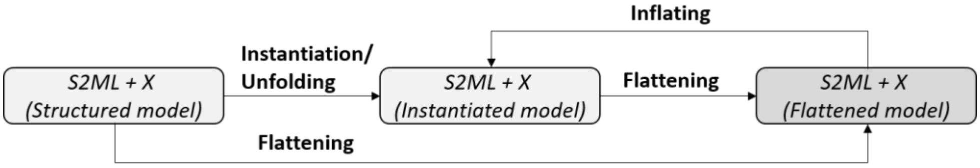

Fig. 2. Operations on S2ML + X models

1) Flattened model

We call hierarchical or structured S2ML model a model made of blocks, instances of classes, and using operations such as cloning, inheritance, composition and aggregation. Any hierarchical or structured S2ML model is semantically equivalent to a flat one, i.e. a model made of a unique block with ports and connections (also called flattened model). The flattened model is obtained by applying recursively rewriting rules, the so-called flattening rules. These rules “remove the walls” of containers (blocks and instances of classes), perform cloning, inheritance and aggregation operations. In the S2ML specification document, they are formally defined in a Structural Operational Semantics style (see e.g. [11]). The operation of transformation of a hierarchical or structured S2ML model into a flattened one is called ***flattening*** (see Fig. 2).

For example, to be assessed by different calculation engines AltaRica 3.0 models are flattened and transformed into Guarded Transition Systems (GTS). This first step makes the assessment more efficient.

2) Instantiated model

Any hierarchical or structured S2ML model is also semantically equivalent to an instantiated or unfolded one, i.e. a model made of a hierarchy of nested/aggregated blocks, connections and ports. In the unfolded model, all the instantiated/inherited classes and “clones” directives are transformed into blocks and all the references/paths to model elements are resolved according to the rewriting rules, the so called unfolding or instantiation rules.
The operation of transformation of a structured S2ML model into an instantiated one is called ***instantiation*** or ***unfolding*** (see Fig. 2). The instantiated S2ML model can be further transformed into a flatten one.

The flattening process can be done in two steps: first, the structured S2ML model is transformed into an instantiated one and then, this model is flattened (see Fig. 2)

For example, the instantiated S2ML model is useful to perform model synchronization, to ensure the consistency between models coming from different engineering domains (see e.g. [12]).

3) Inflating

We call “***inflating***” the operation that produces the opposite effect of "flattening": it creates the instantiated model from the flattened one and reinjects the behaviour into this model (see Fig. 2).
We use this operation reinject the generated stochastic Boolean equations into the original structure of the model.

B. AltaRica 3.0 modelling language

AltaRica 3.0 is a modelling language dedicated to probabilistic safety analyses of complex technical systems [1]. As mentioned previously, AltaRica 3.0 belongs to “S2ML + X” family of modelling languages and is the result of the combination of S2ML (System Structure Modeling Language), a set of object-oriented and prototype-oriented constructs to structure models, and the mathematical framework of GTS (Guarded Transition Systems):

AltaRica 3.0 = S2ML + GTS

1) Guarded Transition Systems

Guarded transition systems belong to the family mathematical models of computation gathered under the generic term of (stochastic) finite-state machines or (stochastic) finite-state automata. They have been introduced in [4] and later refined in [5].

A Guarded Transition system is a quintuple <V, E, T, A, i>, where

- V is a set of state and flow variables,
- E is a set of events, for example, representing failures, repairs of components or system reconfigurations or operator actions;

- T is a set of transitions, each transition is triple <e, G, P>, where e is an event, G is Boolean expression built over V, called a guard, and P is an instruction that modifies the value of state variables and is called post-condition;
- A is an assertion, an instruction to modify the value of flow variables,
- i is a default assignment of state and flow variables.

The internal state of components is represented by state variables. The changes of state are possible when, and only when, an event occurs. The occurrence of an event updates the values of the variables, by the firing of a transition. Dynamic reconfiguration can be represented using transitions.

Flow variables are used to model information circulating between nodes of a model. Their values are calculated from the values of state variables thanks to a mechanism described by means of the so-called assertion. The assertion is executed after each transition firing. Flow variables and assertions make it possible to easily represent failure propagations in the system.

GTS is a compositional modelling formalism, so it is possible to create models of individual components and to assemble them. Probability distributions can associated with events in order to create timed and stochastic models.

2) AltaRica 3.0 model of the tracking system

To create AltaRica 3.0 model of a tracking system given Fig. 1, we first define basic classes. They are defined in Fig. 3.

```
class BasicBlock
  Boolean vsFailed (init = false);
  event evFail_loss (delay = exponential(pLambda));
  parameter Real pLambda = 1.0e-4;

  transition
    evFail_loss: not vsFailed -> vsFailed := true;
end
class BasicInOutBlock
  extends BasicBlock;
  Boolean vfIn, vfOut (reset = false);
  assertion
    vfOut:= vfIn and not vsFailed;
end
class AcquisitionBlock
  extends BasicInOutBlock(pLambda = pAcqLambda);
  parameter Real pAcqLambda = 1.23e-4;
end
class Calculator
  extends BasicBlock (evFail_loss.delay = Weibull(pAlpha, pBeta));
  parameter Real pAlpha = 5.67e+4;
  parameter Real pBeta = 3;

  Boolean vfIn1, vfIn2, vfOut (reset = false);
  assertion
    vfOut := (vfIn1 or vfIn2) and not vsFailed;
end
class Voter
  extends BasicBlock (pLambda = pVoterLambda);
  parameter Real pVoterLambda = 2.64e-7;
  Boolean vfIn1, vfIn2, vfIn3, vfOut (reset = false);
  assertion
    vfOut := (#(vfIn1, vfIn2, vfIn3) >= 2) and not vsFailed;
end
```

Fig. 3. AltaRica 3.0 model of basic components.

We define a class **BasicBlock** that represents the behaviour of basic components. In this class we define a Boolean state variable vsFailed, its value equals to false in the initial configuration. An event **evFail_loss** represents the failure of the component. The probability of occurrence of this event is exponentially distributed with a failure rate given by the parameter **pLambda**. A transition labelled by the event **evFail_loss** defines how changes the state variable **vsFailed**.

Then we define a class **BasicInOutBlock**, which extends **BasicBlock** and adds two Boolean flow variables **vfIn** and **vfOut** and an assertion.

Finally, we define classes **Calculator**, **AcquisitionBlock** and **Voter**. All of them inherits from **BasicBlock** and add some specific behaviour.

The next step is to define the AltaRica 3.0 models of the Acquisition module and LogicSolver. Here we use class/instantiation mechanism to reuse models. We define a class **AcquisitionModule** composed of two instances of the class AquisitionBlock A1 and A2 and two instances of the class Calculator C1 and C2. It also contains an assertion, which represents connections between the components. We also define a class LogicSolver, composed of two instances of the class Voter V1 and V2 and two instances of the class Calculator D1 and D2, an assertion describes connections between these components as given in Fig. 1.

```
class AcquisitionModule
  AcquisitionBlock A1, A2;
  Calculator C1, C2;
  assertion
    C1.vfIn1 := A1.vfOut;
    C1.vfIn2 := A2.vfOut;
    C2.vfIn1 := A1.vfOut;
    C2.vfIn2 := A2.vfOut;
end
```

```
class LogicSolver
  Voter V1, V2;
  Calculator D1 (pAlpha = 3.29+6);
  Calculator D2 (pAlpha = 3.29+6);

  assertion
    D1.vfIn1 := V1.vfOut;
    D1.vfIn2 := V2.vfOut;
    D2.vfIn1 := V1.vfOut;
    D2.vfIn2 := V2.vfOut;
end
```

Fig. 4. AltaRica 3.0 model of the acquisition module and of the logic solver.

We could also use prototype/cloning mechanism instead of class/instance to reuse models.
The model of the whole system is given Fig. 5. It is composed of three instances of the class **AcquisitionModule** M1, M2 and M3, and an instance of the class **LogicSolver** LS. Assertion defines connections between all the components.

```
block TrackingSystem
  AcquisitionModule M1;
  AcquisitionModule M2;
  AcquisitionModule M3;
  LogicSolver LS;

  Boolean S1, S2 (reset = false);

  assertion
    S1 := true;
    S2 := true;
    M1.A1.vfIn := S1;
    M1.A2.vfIn := S2;
    M2.A1.vfIn := S1;
    M2.A2.vfIn := S2;
    M3.A1.vfIn := S1;
    M3.A2.vfIn := S2;
    LS.V1.vfIn1 := M1.C1.vfOut;
    LS.V1.vfIn2 := M2.C1.vfOut;
    LS.V1.vfIn3 := M3.C1.vfOut;
    LS.V2.vfIn1 := M1.C2.vfOut;
    LS.V2.vfIn2 := M2.C2.vfOut;
    LS.V2.vfIn3 := M3.C2.vfOut;

  observer Boolean oFailed = not LS.D1.vfOut and not LS.D2.vfOut;
end
```

Fig. 5. AltaRica 3.0 model of the tracking system (main block)

Once created, this model can be assessed by different tools. In this model we use composition, inheritance and class/instance method to reuse models.
The model given Fig. 5 is a structured model in the sense of S2ML structured model. As explained in the previous section, it is semantically equivalent to an instantiated model given Fig. 6.

```
block TrackingSystem
  block M1
    block A1
```

```
      // behaviour of the block A1
    end
    block A2
      // behaviour of the block A2
    end
    block C1
      // behaviour of the block C1
    end
    block C2
      // behaviour of the block C2
    end
    assertion
      // assertion of the block M1
  end
  block M2
    block A1
      // behaviour of the block A1
    end
    block A2
      // behaviour of the block A2
    end
    block C1
      // behaviour of the block C1
    end
    block C2
      // behaviour of the block C2
    end
    assertion
      // assertion of the block M2
  end
  block M3
    block A1
      // behaviour of the block A1
    end
    block A2
      // behaviour of the block A2
    end
    block C1
      // behaviour of the block C1
    end
    block C2
      // behaviour of the block C2
    end
    assertion
      // assertion of the block M3
  end
  block LS
    block V1
      // behaviour of the block V1
    end
    block V2
      // behaviour of the block V2
    end
    block D1
      // behaviour of the block D1
    end
    block D2
      // behaviour of the block D2
    end
    // assertion of the block LS
  end
  assertion
   // assertion of the block TrackingSystem
end
```

Fig. 6. Instantiated AltaRica 3.0 model of the tracking system

C. New Open-PSA

The new Open-PSA format has been proposed in [9]. It is a result of a combination of S2ML (System Structure Modelling Language) and SBE (Stochastic Boolean Equations), the underlying mathematical formalism of Fault Trees and Reliability Block Diagrams:

New Open-PSA = S2ML + SBE

New Open-PSA provides constructs to represent Stochastic Boolean equations:

- Boolean state variables or basic events (keyword “state” or “basic-event”, they represent basic events of Fault Trees);
- Boolean flow variables or gates (keyword “gate” or “flow”, they represent intermediate events of Fault Trees);
- Probability distributions (they are associated to state variables, for example, exponential, Weibull, etc.);
- Parameters (keyword “parameter”, can be defined and used in the probability distributions);
- Boolean equations (usual logical operators are available, for example and, or, k/n, etc.).

IV. COMPILATION OF ALTARICA 3.0 MODELS INTO BOOLEAN EQUATIONS

A. Compilation algorithm and its improvement

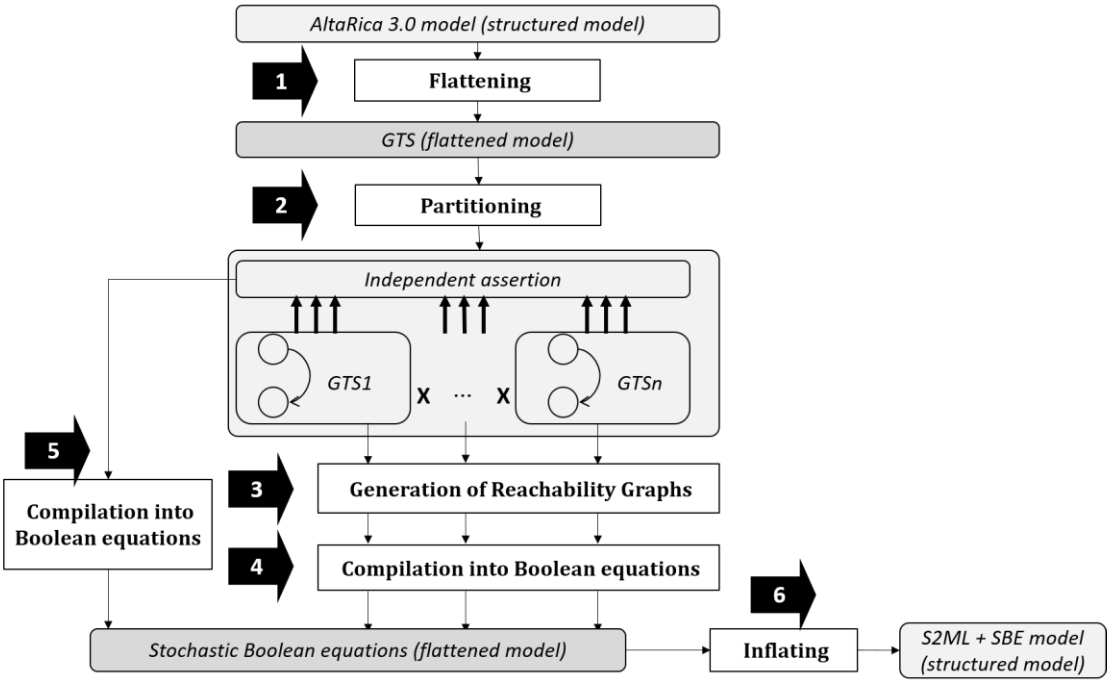

Fig. 7. Compilation algorithm

The initial algorithm to compile AltaRica DataFlow models into Boolean equations has been proposed in [7]. It has been extended to take into account AltaRica 3.0 models with bidirectional flows and loops in the assertion in [8]. Another improvement of the algorithm concerning a more compact representation of generated Boolean equations has been proposed in [10].

The compilation algorithm goes in several steps as illustrated in Fig. 7:

- First, AltaRica 3.0 models are flattened, i.e. transformed into a Guarded Transition Systems (GTS), a model without structure, composed of variables, events, transitions, assertions and initial assignment (see Step 1 of Fig. 7);
- Second, the obtained GTS is partitioned into independent Guarded Transition Systems and an independent assertion (see Step 2 of Fig. 7);
- Third, reachability graphs are generated for the independent guarded transition systems and they are compiled into stochastic Boolean equations (see steps 3 and 4 of Fig. 7);
- Then, the independent assertion is also compiled into stochastic Boolean equations see Step 5 of Fig. 7).

The obtained system of stochastic Boolean equations is a flattened model without any structure as the input guarded transition system. We added a new step in the compilation algorithm (see Step 6 of Fig. 7), which is called “Inflating” and which transforms the flattened stochastic Boolean equations into a structured SBE model. This step produces the opposite effect of "flattening": it reinjects the generated stochastic Boolean equations into the original structure of the model. This makes it possible to obtain a Boolean model close to a hierarchical reliability block diagram, which is directly readable by the analyst.

Finally, we transform an “S2ML+GTS” model into “S2ML + SBE” model. Both models have the same structure.

B. Application to the illustrative example: the tracking system

An extract of stochastic Boolean equations generated from the AltaRica 3.0 model of the Tracking System (see Fig. 5) is given Fig. 8. As you can see, the resulting model is a hierarchical reliability block diagram, which has the same structure as the instantiated AltaRica 3.0 model (see Fig. 6). The behaviour of each block is described by SBE (Stochastic Boolean equations).

```
block TrackingSystem
  gate S1 = true;
  gate S2 = true;
  gate not_oFailed = LS.D1.vfOut or LS.D2.vfOut;
  gate oFailed = LS.D1.not_vfOut and LS.D2.not_vfOut;
  block M1
       block A1
             gate vfIn = owner.owner.S1;
             gate not_vfOut = vfIn and vsFailed;
             gate vfOut = vfIn and not_vsFailed;
             gate not_vsFailed = true;
             gate vsFailed = evFail_loss;
             basic-event evFail_loss = exponential(pLambda, mission-time);
             parameter pLambda = pAcqLambda;
             parameter pAcqLambda = 0.000123;
       end
       block A2
             gate vfIn = owner.owner.S2;
             gate not_vfOut = vfIn and vsFailed;
             gate vfOut = vfIn and not_vsFailed;
             gate not_vsFailed = true;
             gate vsFailed = evFail_loss;
             basic-event evFail_loss = exponential(pLambda, mission-time);
             parameter pLambda = pAcqLambda;
             parameter pAcqLambda = 0.000123;
       end
       block C1
             gate not_vfIn1 = owner.A1.not_vfOut;
             gate vfIn1 = owner.A1.vfOut;
             gate not_vfIn2 = owner.A2.not_vfOut;
             gate vfIn2 = owner.A2.vfOut;
             gate Gate37 = not_vfIn1 and not_vfIn2;
             gate Gate40 = vfIn1 and not_vsFailed;
             gate Gate39 = vfIn2 and not_vsFailed;
             gate not_vfOut = Gate37 or vsFailed;
             gate vfOut = Gate40 or Gate39;
             gate not_vsFailed = true;
             gate vsFailed = evFail_loss;
             basic-event evFail_loss = Weibull(pAlpha, pBeta, mission-time);
             parameter pAlpha = 56700;
             parameter pBeta = 3;
       end
       block C2
             gate not_vfIn1 = owner.A1.not_vfOut;
             gate vfIn1 = owner.A1.vfOut;
             gate not_vfIn2 = owner.A2.not_vfOut;
             gate vfIn2 = owner.A2.vfOut;
             gate Gate42 = not_vfIn1 and not_vfIn2;
             gate Gate45 = vfIn1 and not_vsFailed;
             gate Gate44 = vfIn2 and not_vsFailed;
             gate not_vfOut = Gate42 or vsFailed;
```

```
             gate vfOut = Gate45 or Gate44;
             gate not_vsFailed = true;
             gate vsFailed = evFail_loss;
             basic-event evFail_loss = Weibull(pAlpha, pBeta, mission-time);
             parameter pAlpha = 56700;
             parameter pBeta = 3;
       end
  end
  block M2
    // Stochastic Boolean equations generated for M2(similar to those generated for M1)
  end
  block M3
    // Stochastic Boolean equations generated for M3(similar to those generated for M1)
  end
  block LS
       block V1
             gate not_vfIn1 = owner.owner.M1.C1.not_vfOut;
             gate vfIn1 = owner.owner.M1.C1.vfOut;
             gate not_vfIn2 = owner.owner.M2.C1.not_vfOut;
             gate vfIn2 = owner.owner.M2.C1.vfOut;
             gate not_vfIn3 = owner.owner.M3.C1.not_vfOut;
             gate vfIn3 = owner.owner.M3.C1.vfOut;
             gate Gate11 = not_vfIn2 or not_vfIn3;
             gate Gate14 = not_vfIn1 and Gate11;
             gate Gate12 = not_vfIn2 and not_vfIn3;
             gate Gate17 = vfIn2 and not_vsFailed;
             gate Gate16 = vfIn3 and not_vsFailed;
             gate Gate18 = Gate17 or Gate16;
             gate Gate20 = vfIn1 and Gate18;
             gate Gate19 = vfIn2 and vfIn3 and not_vsFailed;
             gate not_vfOut = Gate14 or Gate12 or vsFailed;
             gate vfOut = Gate20 or Gate19;
             gate not_vsFailed = true;
             gate vsFailed = evFail_loss;
             basic-event evFail_loss = exponential(pLambda, mission-time);
             parameter pLambda = pVoterLambda;
             parameter pVoterLambda = 2.64e-07;
       end
       block V2
       // Stochastic Boolean equations generated for V2 (similar to V1)
       end
       block D1
             gate not_vfIn1 = owner.V1.not_vfOut;
             gate vfIn1 = owner.V1.vfOut;
             gate not_vfIn2 = owner.V2.not_vfOut;
             gate vfIn2 = owner.V2.vfOut;
             gate Gate1 = not_vfIn1 and not_vfIn2;
             gate Gate4 = vfIn1 and not_vsFailed;
             gate Gate3 = vfIn2 and not_vsFailed;
             gate not_vfOut = Gate1 or vsFailed;
             gate vfOut = Gate4 or Gate3;
             gate not_vsFailed = true;
             gate vsFailed = evFail_loss;
             basic-event evFail_loss = Weibull(pAlpha, pBeta, mission-time);
             parameter pAlpha = 3.29 + 6;
             parameter pBeta = 3;
       end
       block D2
       // Stochastic Boolean equations generated for D2 (similar to D1)
       end
  end
end
```

Fig. 8. Stochastic Boolean equations generated from AltaRica 3.0 model of the Tracking system

The Minimal Cut Sets are given in the following table:

Order	MCS
2	LS.V1.evFail_loss LS.V2.evFail_loss LS.D1.evFail_loss LS.D2.evFail_loss
3	LS.V1.evFail_loss M1.C2.evFail_loss M2.C2.evFail_loss

	LS.V1.evFail_loss M1.C2.evFail_loss M3.C2.evFail_loss LS.V1.evFail_loss M2.C2.evFail_loss M3.C2.evFail_loss M2.C1.evFail_loss M3.C1.evFail_loss LS.V2.evFail_loss M1.C1.evFail_loss M2.C1.evFail_loss LS.V2.evFail_loss M1.C1.evFail_loss M3.C1.evFail_loss LS.V2.evFail_loss
4	M1.A1.evFail_loss M1.A2.evFail_loss M2.A1.evFail_loss M2.A2.evFail_loss M1.A1.evFail_loss M1.A2.evFail_loss M2.C1.evFail_loss M2.C2.evFail_loss M1.A1.evFail_loss M1.A2.evFail_loss M3.C1.evFail_loss M2.C2.evFail_loss M1.A1.evFail_loss M1.A2.evFail_loss M3.A1.evFail_loss M3.A2.evFail_loss M1.A1.evFail_loss M1.A2.evFail_loss M2.C1.evFail_loss M3.C2.evFail_loss M1.A1.evFail_loss M1.A2.evFail_loss M3.C1.evFail_loss M3.C2.evFail_loss M1.A1.evFail_loss M1.A2.evFail_loss M2.C1.evFail_loss LS.V2.evFail_loss M1.A1.evFail_loss M1.A2.evFail_loss M3.C1.evFail_loss LS.V2.evFail_loss M1.C1.evFail_loss M2.A1.evFail_loss M2.A2.evFail_loss M3.C2.evFail_loss M1.C1.evFail_loss M3.A1.evFail_loss M3.A2.evFail_loss M2.C2.evFail_loss M1.C1.evFail_loss M2.C1.evFail_loss M2.C2.evFail_loss M3.C2.evFail_loss M1.C1.evFail_loss M3.C1.evFail_loss M2.C2.evFail_loss M3.C2.evFail_loss M1.C1.evFail_loss M3.A1.evFail_loss M3.A2.evFail_loss M1.C2.evFail_loss M1.C1.evFail_loss M2.A1.evFail_loss M2.A2.evFail_loss M1.C2.evFail_loss M1.C1.evFail_loss M2.C1.evFail_loss M1.C2.evFail_loss M2.C2.evFail_loss M1.C1.evFail_loss M3.C1.evFail_loss M1.C2.evFail_loss M2.C2.evFail_loss M1.C1.evFail_loss M2.C1.evFail_loss M1.C2.evFail_loss M3.C2.evFail_loss M1.C1.evFail_loss M3.C1.evFail_loss M1.C2.evFail_loss M3.C2.evFail_loss M1.C1.evFail_loss M2.A1.evFail_loss M2.A2.evFail_loss LS.V2.evFail_loss M1.C1.evFail_loss M3.A1.evFail_loss M3.A2.evFail_loss LS.V2.evFail_loss M2.A1.evFail_loss M2.A2.evFail_loss M3.A1.evFail_loss M3.A2.evFail_loss M2.A1.evFail_loss M2.A2.evFail_loss M3.C1.evFail_loss M3.C2.evFail_loss M2.A1.evFail_loss M2.A2.evFail_loss M3.C1.evFail_loss M1.C2.evFail_loss M2.A1.evFail_loss M2.A2.evFail_loss M3.C1.evFail_loss LS.V2.evFail_loss M2.C1.evFail_loss M3.A1.evFail_loss M3.A2.evFail_loss M2.C2.evFail_loss M2.C1.evFail_loss M3.A1.evFail_loss M3.A2.evFail_loss M1.C2.evFail_loss M2.C1.evFail_loss M3.A1.evFail_loss M3.A2.evFail_loss LS.V2.evFail_loss M2.C1.evFail_loss M3.C1.evFail_loss M2.C2.evFail_loss M3.C2.evFail_loss M2.C1.evFail_loss M3.C1.evFail_loss M1.C2.evFail_loss M2.C2.evFail_loss M2.C1.evFail_loss M3.C1.evFail_loss M1.C2.evFail_loss M3.C2.evFail_loss M2.A1.evFail_loss M2.A2.evFail_loss LS.V1.evFail_loss M3.C2.evFail_loss M3.A1.evFail_loss M3.A2.evFail_loss LS.V1.evFail_loss M2.C2.evFail_loss M1.A1.evFail_loss M1.A2.evFail_loss LS.V1.evFail_loss M2.C2.evFail_loss M1.A1.evFail_loss M1.A2.evFail_loss LS.V1.evFail_loss M3.C2.evFail_loss M3.A1.evFail_loss M3.A2.evFail_loss LS.V1.evFail_loss M1.C2.evFail_loss M2.A1.evFail_loss M2.A2.evFail_loss LS.V1.evFail_loss M1.C2.evFail_loss

V. CONCLUSION AND PERSPECTIVES

AltaRica 3.0 is a modelling language dedicated to probabilistic safety analyses of complex technical systems. It is integrated in AltaRica 3.0 Workshop, a modelling environment that provides several tools for processing AltaRica 3.0 models: an interactive simulator, a stochastic simulator, a generator of critical sequences as well as a compiler to stochastic Boolean equations.

In this article, we presented the improvement of this compilation algorithm. We added a new step to the compilation process. This step, which could be called "inflating", produces the opposite effect of "flattening": it reinjects the generated stochastic Boolean equations into the original structure of the model. This makes it possible to obtain a Boolean model close to a hierarchical reliability block diagram, which is directly readable by the analyst.

This new compilation algorithm reinforces the AltaRica 3.0 technology and thereby the so-called model-based approach for dependability analyses.

REFERENCES

[1] M. Batteux, T. Prosvirnova & A. Rauzy. AltaRica 3.0 in ten modelling patterns. International Journal of Critical Computer-Based Systems. Inderscience Publishers. Vol. 9, Num. 1-2, pp 133-165, 2019

[2] Michel Batteux, Tatiana Prosvirnova and Antoine Rauzy. From Models of Structures to Structures of Models. IEEE International Symposium on Systems Engineering (ISSE 2018). IEEE. Roma, Italy. October, 2018. doi:10.1109/SysEng.2018.8544424. Best paper award

[3] Antoine Rauzy and Cecilia Haskins. Foundations for Model-Based Systems Engineering and Model-Based Safety Assessment. Journal of Systems Engineering. Wiley Online Library. 22. pp. 146–155. 2019. doi:10.1002/sys.21469.

[4] Antoine Rauzy. Guarded Transition Systems: a new States/Events Formalism for Reliability Studies. Journal of Risk and Reliability. Professional Engineering Publishing. 222:4. pp. 495–505. 2008. doi:10.1243/1748006XJRR177.

[5] Michel Batteux, Tatiana Prosvirnova and Antoine Rauzy. AltaRica 3.0 Assertions: the Why and the Wherefore. Journal of Risk and Reliability. Professional Engineering Publishing. 231:6. pp. 691-700. September, 2017. doi:10.1177/1748006X17728209.

[6] Antoine Rauzy. Probabilistic Safety Analysis with XFTA. AltaRica Association. Les Essarts le Roi, France. ISBN 978-82-692273-0-7. 2020.

[7] Antoine Rauzy. Modes Automata and their Compilation into Fault Trees. Reliability Engineering and System Safety. Elsevier. 78:1. pp. 1–12. October, 2002. doi:10.1016/S0951-8320(02)00042-X.

[8] Tatiana Prosvirnova and Antoine Rauzy. Automated generation of Minimal Cutsets from AltaRica 3.0 models. International Journal of Critical Computer-Based Systems. Inderscience Publishers. 6:1. pp. 50–79. 2015. doi:10.1504/IJCCBS.2015.068852.

[9] Michel Batteux, Tatiana Prosvirnova & Antoine Rauzy, The New Open-PSA Format: a Model-Based Approach, In Actes du congrès Lambda-Mu 22 (actes électroniques), IMdR. Le Havre, France. October, 2020.

[10] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy, Advances in the simplification of Fault Trees automatically generated from AltaRica 3.0 model, In Stein Haugen and Anne Barros and Coen van Gulijk and Trond Kongsvik and Jan Erik Vinnem ed., *Safe Societies in a Changing World, proceedings of European Safety and Reliability Conference (ESREL 2018)*. Trondheim, Norway. pp 907–914, June, 2018.

[11] Michel Batteux, Tatiana Prosvirnova, Antoine Rauzy. System Structure Modeling Language (S2ML). 2015. ⟨hal-01234903⟩

[12] Michel Batteux, Jean-Yves Choley, Faïda Mhenni, Tatiana Prosvirnova & Antoine Rauzy, Synchronization of system architecture and safety models: a proof of concept, In Proceedings of the IEEE 2019 International Symposium on Systems Engineering (ISSE). Edinburgh, Scotland. October 2019.