www.acsysteme.com/fr/lmcs

LMCS 2015

Logiciels pour la modélisation et le calcul scientifique

mardi 24 novembre 2015, site de l'Inria à Rennes (35), France

Conférencier : Marc BUISSOU

Organisme : EDF

Modelica et sûreté de fonctionnement

Contexte du projet

Abstract: Modelica is a modeling language which was created in order to ease the description of multi physics systems thanks to an object oriented approach. Modelica models usually represent only the nominal functioning of systems and are used to simulate them for design purposes. This article proposes various ways to derive dependability models from such Modelica models, as automatically as possible. Depending on the tightness of coupling between the continuous processes and discrete events such as failures and repairs, the methods proposed here range from the addition of stochastic behavior in the Modelica model itself to the association of the system structure to a library written in the Figaro reliability modeling language. Such an association allows the use of the Figaro workbench tools, specially designed for dependability analysis; it is then possible to generate a fault tree, or to create automatically a discrete stochastic simulation model. All these possibilities are exemplified through two classical use cases: a telecommunication network, and a "heated tank" system.

Keywords: Modelica, reliability analysis, hybrid stochastic system, fault tree generation.

From Modelica models to dependability analysis

M. Bouissou*' **. X. de Bossoreille***

**Electricité de France, R&D, Clamart, 92141*
France (marc.bouissou@edf.fr)
***Ecole Centrale Paris, Chatenay Malabry, 92295*
France (marc.bouissou@ecp.fr)
**** DLR, Institute of System Dynamics and Control, D-82234 Wessling,*
Germany (xavier.bossoreille@dlr.de)

Abstract: Modelica is a modeling language which was created in order to ease the description of multi physics systems thanks to an object oriented approach. Modelica models usually represent only the nominal functioning of systems and are used to simulate them for design purposes. This article proposes various ways to derive dependability models from such Modelica models, as automatically as possible. Depending on the tightness of coupling between the continuous processes and discrete events such as failures and repairs, the methods proposed here range from the addition of stochastic behavior in the Modelica model itself to the association of the system structure to a library written in the Figaro reliability modeling language. Such an association allows the use of the Figaro workbench tools, specially designed for dependability analysis; it is then possible to generate a fault tree, or to create automatically a discrete stochastic simulation model. All these possibilities are exemplified through two classical use cases: a telecommunication network, and a "heated tank" system.

Keywords: Modelica, reliability analysis, hybrid stochastic system, fault tree generation.

1. INTRODUCTION

The purpose of reliability, and more generally, of dependability studies is to calculate probabilities of undesirable events such as the failure of the mission of a system, or to estimate the probability distribution of some performances of the system: total production on a given time interval, maintenance cost, number of repairs etc. Usually, dependability studies are performed with dedicated methods and tools, based on discrete (and often even Boolean) models of systems: fault trees, Markov chains, Petri nets, BDMP (Boolean logic Driven Markov Processes) etc. EDF (Electricité de France) has a lot of experience about reliability modelling, and designed the Figaro modelling language in 1990. This language generalizes all the above cited models, and allows casting knowledge on categories of systems in libraries. It is the basis of KB3 which is the reference tool used for building fault trees and dynamic models for probabilistic safety analyses of nuclear power plants and all other reliability analyses at EDF.

Unfortunately, in some situations, a purely discrete representation of a system cannot be a good enough approximation: this is the case of hybrid systems, having both discrete and continuous parts, with strong interactions between them. Reliability analysts call the study of such systems "dynamic reliability". For such systems, the Figaro language can only provide step function approximations for representing the evolution of continuous variables: cf. Bouissou (2007). This is why we looked for better solutions for hybrid systems, and investigated the use of Modelica. In Bouissou et al. (2014) we showed that it is possible to add stochastic behaviour due to failures and repairs of components to a deterministic hybrid system described in Modelica. This paper explains how one can perform Monte Carlo simulations on such models, with a smart algorithm that minimizes the number of random numbers that must be generated. Moreover, this algorithm can be implemented using the standard Modelica solvers, which is a major advantage. We also looked for solutions to perform more classical dependability analyses (starting from a Modelica model) for systems for which the hybrid behaviour can be abstracted into a discrete behaviour, or even a Boolean model such as a fault tree. Our conclusion was that using Modelica itself would be extremely difficult because it would imply the cohabitation of two models in one: a detailed, physical simulation model for the nominal functioning and a more abstract model suited for dependability analyses. We finally concluded that a much easier and more efficient solution would be to generate automatically a Figaro model from the Modelica model, then use the mature Figaro tools for performing the needed transformations (like the generation of a fault tree) and calculations (of availability, reliability etc.).

The main contribution of the present paper is to describe a method for doing this. This method is already partially (a few manual inputs are still necessary) implemented in two prototypes based respectively on the tools Dymola and

OpenModelica, associated in both cases to the tools of the Figaro workbench.

An auxiliary contribution of the present paper is the application of the method described in Bouissou et al. (2014) to a more complicated use case, which has served as a benchmark for numerous other methods. This application shows two things: the model is simple and “natural” with the proposed approach and the performances of the simulation are better than those of a recently published paper.

To summarize, this paper illustrates by examples some of the results related to dependability analysis which were obtained in the MODRIO European project. It is organized as follows: the largest part of the article (sections 2 to 4) presents the connection that can be established between detailed simulation models in Modelica, and more abstract models written in Figaro, in order to generate fault trees or to perform probabilistic evaluations based on discrete models. Section 5 is an example of such a connection (for a telecom network), and Section 6 compares the method described in Bouissou et al. (2014) to other techniques on a well-known benchmark.

2. THE FIGARO LANGUAGE AND TOOLS

Given the existence of a large community of Modelica users, we will assume that this language is already known by the reader. The dedicated web site Modelica.org provides thousands of free resources including articles, component libraries, tutorials, forums, links to open source and commercial tools, and of course the detailed specifications of the Modelica language. The Figaro language is more specific, this is why we give its main characteristics here.

The Figaro language, created in 1990, is a domain specific object oriented modelling language dedicated to dependability with the following objectives, commented and exemplified in Bouissou et al. (1991):

- provide an appropriate formalism for developing libraries (with generic descriptions of components);
- be more general than all the usual reliability models. For example, in the above cited paper, it is shown that reliability block diagrams and Petri nets can be represented in Figaro;
- find the best trade-off between modelling power (or generality) and possibilities for the processing of models. In particular, models with differential equations have been explicitly excluded from the scope of Figaro;
- be as legible as possible (see example in section 5);
- be easily associated with graphic representations. In (Bouissou et al. 1991) one can see how Petri nets, reliability block diagrams and also an electrical system could be input with their usual graphical representations in the very first version of the KB3 tool, based on Figaro descriptions.

The Figaro language has two levels, called order 1 and order 0. At order 1, its syntax allows to define generic constructions contained in reusable classes, while at order 0 it can only describe a particular system by means of objects.

A class consists of two parts:

⇒ a purely **static** and declarative part :

- name of the class and of the class(es) whose characteristics it inherits;
- interfaces (classes with which the concerned class will interact, possibly with constraints on the number of objects authorised for each interface);
- constant characteristics;
- state variables, with their initial values.

⇒ a **dynamic** part : the occurrence and interaction rules describing the behaviour of the class. The occurrence rules describe elementary events with the conditions governing how they are triggered and the associated probability distributions. The purpose of the interaction rules is to propagate the effects that are immediate and certain consequences of an event in the system. These rules often make use of quantifiers in order to be valid irrespective of the content of sets of objects defined by the interfaces; simple examples of the use of quantifiers are given in section 5.3 (cf. class "link").

The tool KB3 was designed to offer a generic graphical user interface for working with Figaro models, that can be tailored to each Figaro library. In KB3, a Figaro class can be associated equally well with an icon as with a link. This means that a Figaro link can be a complex object with rules, and not just a means to declare constraints (equality, conservation of flow) on state variables. This is an important difference between Figaro and Modelica.

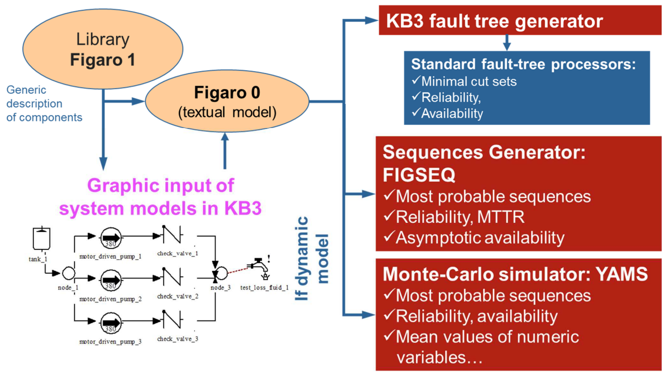

Fig. 1. The Figaro workbench overall architecture.

Once the architecture of a system has been graphically input in KB3, the man-machine interface translates it into a list of objects described in Figaro language. The set "library + list of parameterized objects" is a complete model in order 1 Figaro language for a given system. This model is concise, but it would be very complex to use it directly, and not all the recommended checks could be run on it. For this reason, prior to any processing, this model is fully instantiated in order 0 Figaro, a very simple sub-language of order 1 Figaro which is suitable for description of the behaviour of a particular system, and which enables all consistency checks to be run and effective processing to be carried out. A formal definition of the semantics of the order 0 Figaro language is given in Bouissou and Houdebine (2002).

The Figaro workbench is a set of tools designed to help a user define Figaro models, then process them in order to perform dependability analyses. Fig. 1 gives an overview of the main tools of the workbench, and shows how they are connected.

3. ENGINEERING WORKFLOWS

At EDF there are traditionally two parallel workflows for design and for dependability analyses, in different departments. In the first workflow, designers build Modelica models and simulate them in order to optimize the normal operation of systems; in the second one, reliability engineers estimate non-functional performances such as reliability, availability and maintainability of the system.

The design information, needed to perform dependability studies, is "manually" transferred from the first to the second workflow. With the approach developed in the MODRIO project, it is now possible to transfer automatically the system structure from the first to the second workflow. This is illustrated on Fig. 2, which emphasizes fault tree based dependability studies; of course other studies of Fig. 1 are also possible.

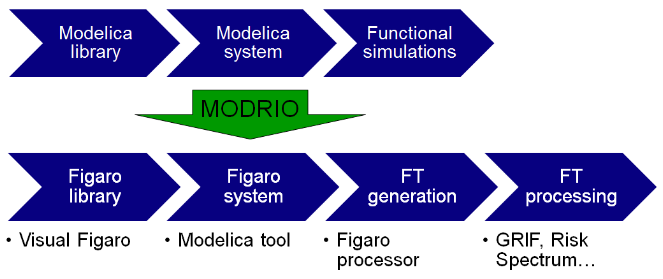

Fig. 2. The design and dependability analyses workflows at EDF.

The benefits of this automatic transfer are:

- assurance of consistency between the models used in the two workflows;
- time saving, making it possible to get an immediate feedback from dependability performances during the design process;
- the workflows remain largely independent; in particular, the Modelica and Figaro libraries can still be developed independently. Even the tools used in the two workflows can be maintained independently, except for their interface with the bridge created between them.

4. FROM MODELICA TO FIGARO

Fig. 3 explains the principles used to switch from a Modelica model to a Figaro model. Starting from a pure Modelica model designed for physical simulation, one can obtain a Figaro model by extracting objects (maybe not all of them, see example 1 below) and their inter-relations from the model and associating them to a well suited library in Figaro.

The Figaro library contains the dependability models, including in particular the failures and repairs of components, with the associated reliability data (default values of failure rates and mean repair times).

There are two possible levels of automation in the process.

The first level (the one that is currently implemented in the two prototypes) is library independent, but it requires a few inputs from the user. In fact the binding of objects between the Modelica and the Figaro model must be "loose", because:

- the structure of the Figaro model may be quite different from the structure of the Modelica model. This is due to the fact that different kinds of abstraction are used when going from a real system to a functional, simulation model on the one hand, and to a dysfunctional model on the other hand;
- different Figaro libraries can be bound to a single Modelica model; this gives the possibility to do various kinds of studies, all from a single initial model.

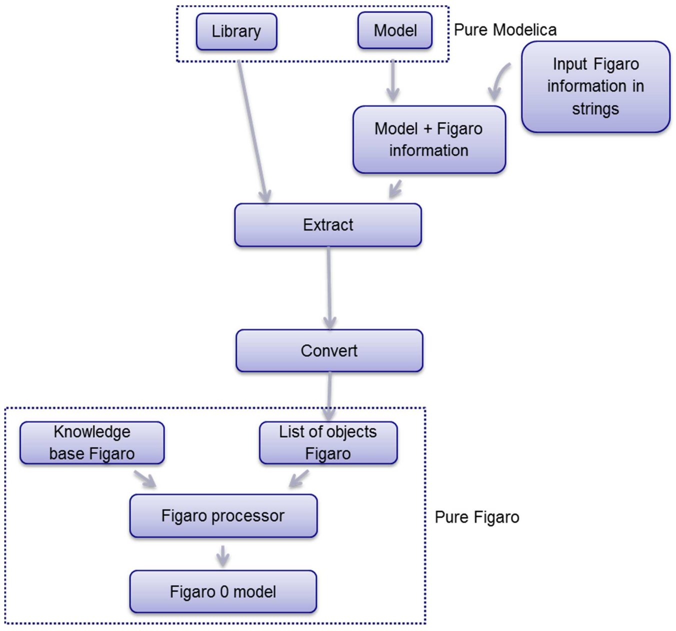

Fig. 3. From Modelica to Figaro.

The binding is specified by associating a Figaro class name to each object of the Modelica model and by declaring the inter-relations between objects, directly in Figaro syntax. This second kind of information is necessary because, as mentioned earlier, the ways connections between components are declared are quite different in Modelica and Figaro. This is exemplified in the telecom example in the next section.

In order to generate Figaro 0 models from a Modelica tool, all the parameters and information needed by Figaro and that do not exist in Modelica must be integrated in Modelica in such a way that they do not affect in any way the original simulation model (without Figaro). That is why they must be added as annotations or as string parameters. We decided to use parameters, because they can be inherited, and this is very important in our approach. Moreover, parameters are part of the core of the Modelica language, which means that it will be possible to use the same models with *all* Modelica tools.

The second level of automation consists in filling automatically the information on connections in the strings added to the Modelica model. This is library dependent, and we will explain how it can be done on the example given in the next section.

5. EXAMPLE 1 (DISCRETE): TELECOM NETWORK

This example will probably seem a bit artificial: this is due to the fact that we started from the abstract representation (explained in section 5.1) to derive a physical model (section 5.2) while on real systems it is the other way round! But we chose that example because it is simple enough to be explained in an article and yet it poses difficult problems. We have reported in a MODRIO document (Bouissou 2014) how the same principles work perfectly on a real thermohydraulic system of a nuclear power plant with libraries encompassing thousands of lines in Modelica, and hundreds of lines in Figaro.

5.1 Telecommunication network

In a telecommunication network such as the one represented in Fig. 4, the classical so-called S-T connectivity problem consists in calculating the probability that a given target (a blue node) is connected to at least one source of information (a green node). Here, we make the simplest possible assumptions on the failure and repair processes of components: failure and repair times are all exponentially distributed, and components are all independent. Nodes and links can both fail. Despite the apparent simplicity of this example, it poses real challenges for generating fault trees: there are loops in the topology of the system, which, without a proper treatment will lead to invalid fault trees (containing loops). Moreover, the number of links connected to a node is not known in advance and thus the reliability model must be written in a way which is independent from the number of connections. To the best of our knowledge, these two problems are solved *only* by the tools based on Figaro.

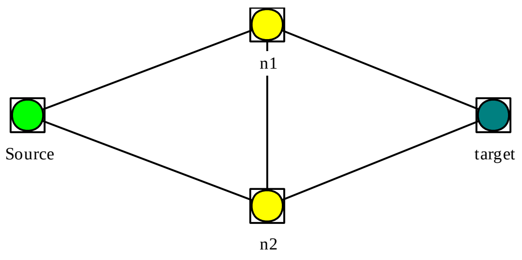

Fig. 4. Telecom network.

5.2 Representation as a hybrid system

A possible analog model of such a network in Modelica is an electrical circuit (built using the Modelica.Electrical.Analog library) where links are represented by resistors and source nodes by generators; the other nodes are represented by pins (components without behaviour that just serve as "hubs" for connections). Since we want to detect the propagation of the alternative signal generated by the source, three auxiliary resistors of 1 Ohm are needed. They do not have Figaro string parameters; hence they disappear completely in the conversion from the Modelica + Figaro strings model to the pure Figaro model. This shows an example of management of structural differences between the Modelica and the Figaro model.

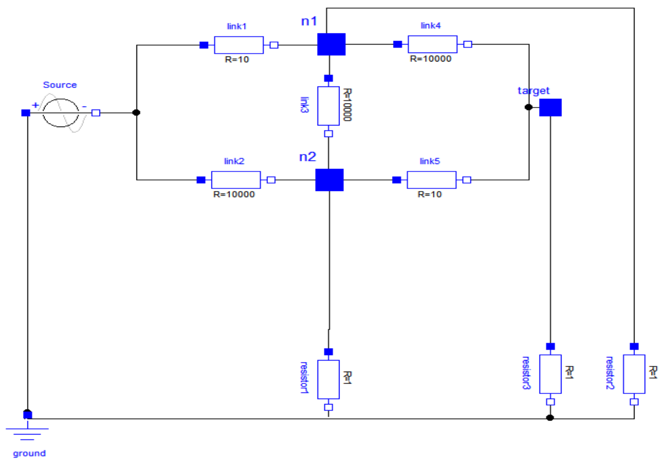

Fig. 5. Analog model of a telecom network: electrical circuit.

Each link is represented by a resistor. Its normal behaviour is represented by a value of 10 Ohms, whereas its failure is represented by a value of 10,000 Ohms. It is also possible to simulate a failure of a node by setting the corresponding auxiliary resistor to 0.

If only the first level of automation for going from the Modelica to the Figaro model is used, the user has to input manually the following information in the five resistors considered as links: "INTERFACE extremity x y;" where x and y are the nodes connected to the two ports of the link. This minimal information can be completed, if needed, by instructions declaring values for the failure or repair rates that will override the default values defined in the Figaro library.

In order to leverage automation to level 2, one would have to write a little algorithm in Modelica that would do the following:

```
For X in objects of the model
   If X of class Link
    Write in X.figaro_string : "INTERFACE ",
     name of object connected to X.positive_pin,
     name of object connected to X.negative_pin, ";"
   End If
End for
```

5.3 The Figaro knowledge base for telecom networks

Here is *in extenso* the library needed to generate a fault tree for *any topology* of telecom network that could be deduced from a Modelica simulation model such as the one of Fig. 4. Figaro keywords are written in uppercase.

```
CLASS node;
 CONSTANT
 function DOMAIN 'source' 'target' 'intermediate'
         DEFAULT 'intermediate';
 lambda DOMAIN REAL DEFAULT 1e-5;
 mu DOMAIN REAL DEFAULT 0.1;
 FAILURE fail LABEL "Failure of %OBJECT"
      RELIABILITY_DATA MODEL_GLM
       GAMMA 0.
       LAMBDA lambda
       MU mu;
 EFFECT connected
        LABEL "%OBJECT is linked to a source";
 INTERACTION
   IF WORKING AND function = 'source'
   THEN connected;
```

```
CLASS source KIND_OF node ;
 CONSTANT function DEFAULT 'source';

CLASS target KIND_OF node ;
 CONSTANT function DEFAULT 'target';

CLASS link ;
 INTERFACE extremity KIND node CARDINAL 2 ;
 FAILURE interruption LABEL "The link %OBJECT is
        broken"
      RELIABILITY_DATA MODEL_GLM
       GAMMA 0.
       LAMBDA 1e-5
       MU 0.1 ;
 INTERACTION
   IF WORKING AND
    (FOR_ANY x AN extremity WE_HAVE WORKING(x))
     AND IT_EXISTS x AN extremity SUCH_THAT
         connected OF x
   THEN FOR_ALL z AN extremity DO connected(z);
```

5.4 Fault tree generation

Once the Figaro model is available (library + objects automatically deduced from the Modelica model), the simple invocation of the Figaro processor produces an XML file containing a fault tree, which can be represented graphically as in Fig. 6.

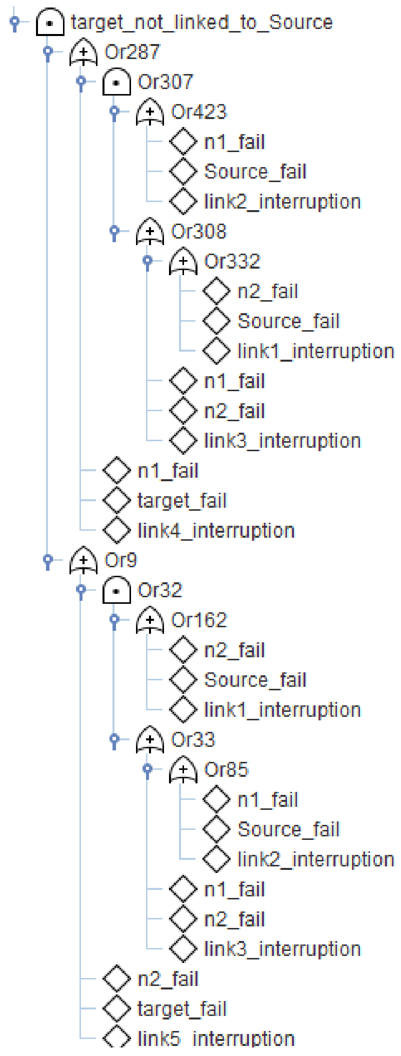

Fig. 6. Fault tree generated from the telecom network.

5.5 Fault tree analysis

The fault tree produced by the Figaro processor can then be transferred (maybe via an ad hoc translator) to any fault tree processing tool. With such a tool, different kinds of results can be obtained: minimal cut sets, the probability of the top event, importance factors of components…

This is not available in the prototype tools, but one can imagine that some of the results of the fault tree analysis can be used in conjunction with the Modelica models. For example, a minimal cut set could be transformed into a script that could be run on the Modelica model, in order to "check" that this combination of failures indeed leads to a failure of the system.

6. EXAMPLE 2 (HYBRID): HEATED TANK

6.1 History of this benchmark

The heated-tank problem was first introduced in Aldemir (1987), and since then has been serving as a benchmark for dynamic reliability. It has been solved with different approaches including Marseguerra et. al. (1995), Tombuyses et. al. (1996), Lair et. al. (2010), Zhang et. al. (2009), Zhang et. al. (2013). This problem is not trivial because it contains two coupled continuous variables (fluid level and temperature), and the components failure rates heavily depend on the temperature. This example has the advantage of being defined with much precision and being sufficiently small to enable exhaustive comparison of different solving approaches.

6.2 System short description

Due to space limitations, we limit the description to its main features. Numerical parameters can be found in Zhang et al. (2013) and several other references.

The main component of the system is a tank containing a fluid. Two pumps (components 1 and 2) can add fluid in the tank. A valve (component 3) can remove fluid from the tank. The pumps and the valve can each be either ON or OFF; they are controlled by level sensors. The pumps and the valve can present failures, leading them to be either STUCK_ON or STUCK_OFF. The times to apparitions of these failures are governed by the integration of the failure rates, according to the following formula, where T is the (random) time to failure:

$$\Pr(\mathrm{T} \le t) = 1 - \exp(-\int_0^t \lambda(\theta(\mathrm{u}))du)$$

The failure rates *λ(θ)* vary with the temperature of the fluid. Lastly, a heating device heats the fluid in the tank.

The components are controlled according to two laws:

⇒ if the fluid level *h(t)* drops below 6m, the components 1, 2, 3 are put respectively in state ON, ON, OFF (assuming they are not STUCK_ON or STUCK_OFF)

⇒ if the fluid level *h(t)* rises above 8m, the components 1, 2, 3 are put respectively in state OFF, OFF, ON (assuming they are not STUCK_ON or STUCK_OFF)

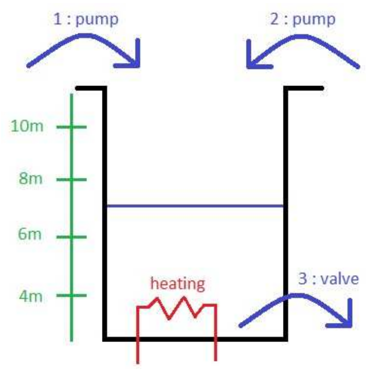

Fig. 7. The "heated tank" system.

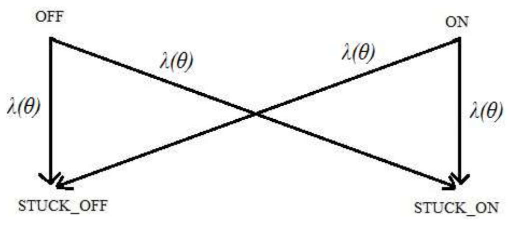

Fig. 8. States and transitions of the components.

The two continuous-variables are the fluid level *h(t)* and the fluid temperature *θ(t)*. They satisfy the following differential equations:

$$\mathrm{dh/dt} = (v_1 + v_2 - v_3)F/A$$
$$h\,d\theta/dt = (v_1 + v_2)F(T_p - \theta)/A + Q/A$$
with
$$v_c = \begin{cases} 0 \text{ if c is OFF or STUCK\_OFF} \\ 1 \text{ if c is ON or STUCK\_ON} \end{cases}, c \in \{1,2,3\}$$
$F, T_p, A,$ and Q constants

We consider the system to be failed if it reaches either of the following situations: drought (*h*<4m), overflow (*h*>10m) or boiling (*θ*>100°C). We are interested in the probabilities of these events occurring before time *t*.

6.3 Model in Modelica

We implemented this system with two different modelling methods: state-graphs, using a modified version of the Stategraph2 library (Otter et al. 2009) and state-machines (this is the most elegant, and it is detailed below). Both use the mechanisms given in Bouissou et al. (2014) for the generation of random events and the Monte Carlo simulation. Thanks to the features of Modelica, these mechanisms can be cast in a library and hidden from the user who builds models.

In the following, we explain all what the user has to input to solve this problem, supposing that he has this library at hand.

The top level of the system model is shown in figure 9. The Tank only contains the two differential equations and their initial conditions. The Control contains a state machine (purely deterministic) with 3 states: High, Intermediate, Low. The transitions between states are triggered by comparisons between the current fluid level and the various thresholds. According to the active state, commands are sent to pumps and the valve.

The most interesting components are Pump_1, Pump_2 and Valve, since they are the only components with a random behaviour.

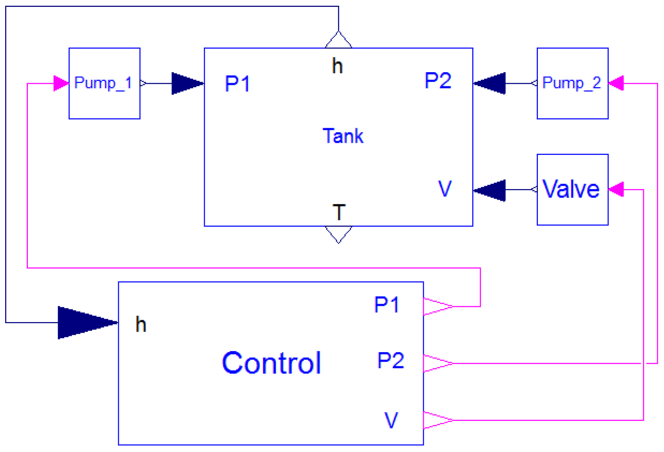

Fig. 9. Structure of the Modelica model.

They contain the state-machine depicted in Fig.10. Deterministic continuous-time state-machines in Modelica were already presented in Elmqvist et al. (2014). We used them in order to implement stochastic transitions in accordance with the principles explained in Bouissou et al. (2014).

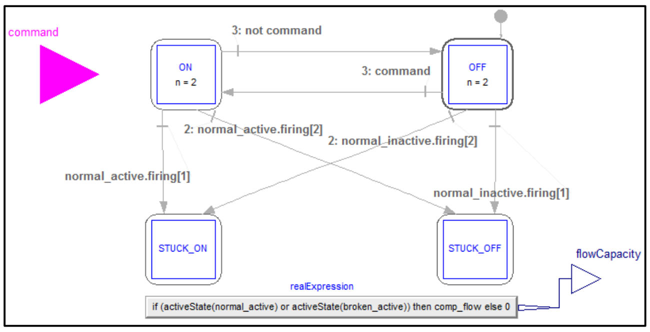
command
3: not command
ON
n = 2
3: command
OFF
n = 2
2: normal_active.firing[2]
2: normal_inactive.firing[2]
normal_active.firing[1]
normal_inactive.firing[1]
STUCK_ON
STUCK_OFF
flowCapacity
realExpression
if (activeState(normal_active) or activeState(broken_active)) then comp_flow else 0

Fig. 10. Continuous-time state-machine model for the two pumps and the valve.

This state-machine contains both deterministic transitions (linked to control laws) and stochastic transitions. Moreover, the failure rates *λ(θ)* vary with the temperature calculated in the tank.

This whole model cannot give us directly the desired cumulated probabilities of failures. One simulation can only give us *one* possible trajectory, determined by the random numbers generated for this simulation. This is why we do Monte-Carlo simulations: we launch a large number of simulations with different seeds for the pseudo random number generator. The mean of the results then converges towards the sought result.

6.4 Results and comparison with other articles

As stated before, this benchmark has been studied in many different ways in the past. We will here compare our results with the ones in Zhang et al. (2009) and the ones in Zhang et al. (2013). In Zhang et al. (2009), the authors use an analytical solution to solve differential equations. This way, they manage to simulate 10^5 histories in 1min37s. Compared to them, we do not seem to be efficient with 1hour. However, contrary to them, we do not use an analytical solution, which would require first mathematically studying the system. We only use the original raw equations and let Dymola do the numerical integration. This explains the difference in speed.

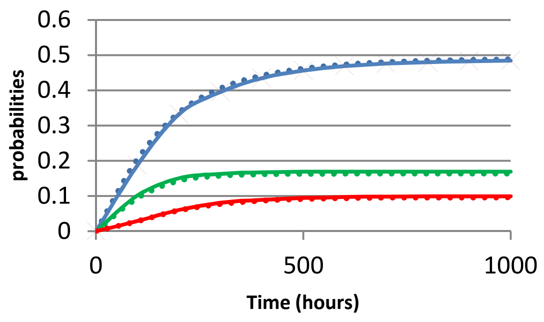

Fig. 11. Cumulated probabilities of failures. Comparison between our results (105 histories) and PDMP from Zhang et al. (2009) (107 histories, dotted). Blue: overflow; Green: boiling; Red: drought.

In fact, in Zhang et al. (2013), the authors try this time to use a more general method to solve this benchmark with Matlab/Simulink. In this case, simulating 10^5 histories takes them about 23hours (versus 1hour for us). This difference is in part due to them having to use a fixed-step integration algorithm. With our method, explained in Bouissou et al. (2014), the integration uses a variable-step algorithm and is thus more efficient.

6. CONCLUSION

Thanks to the MODRIO project, a bridge was built between the design and dependability analysis workflows for complex systems. Starting from a Modelica model, it is now possible to generate a fault tree or a discrete stochastic simulation model thanks to an automatic transfer of information towards mature dependability tools based on the Figaro modelling language. For hybrid systems, we have developed another approach, based on the introduction of stochastic behaviour in the Modelica model itself. This approach compares well to the state of the art in terms of modelling effort and simulation time.

The two modelling methods exemplified here on well-known use cases could easily be applied to much larger and more complex systems, saving a lot of modelling time, while using state of the art dependability analysis techniques.

AKNOWLEDGEMENTS

This paper is based on research performed within the ITEA2 project MODRIO. Partial financial support of the French DGE for this development is highly appreciated.

REFERENCES

Aldemir T. (1987). Computer-Assisted Markov Failure Modeling of Process Control Systems. IEEE *Trans. on Reliability*, volume 36(4), pages 133-144.

Bouissou M., Bouhadana H., Bannelier M., Villatte N. (1991). Knowledge modelling and reliability processing: presentation of the FIGARO language and associated tools. Safecomp'91, Trondheim (Norway).

Bouissou M., Houdebine (2002). Inconsistency detection in KB3 models. ESREL 2002, Lyon (France), March.

Bouissou M. (2007). Comparison of two Monte Carlo schemes for simulating Piecewise Deterministic Markov Processes. MMR07, Glasgow, (UK), July 2007.

Bouissou M. (2014). Specification of Modelica extensions and interfaces for Bayesian networks and Fault trees. Deliverable D.2.2.1 of the MODRIO Artemis project.

Bouissou M., Elqmvist H., Otter M., and Benveniste A. (2014). Efficient Monte Carlo simulation of stochastic hybrid systems. Modelica'2014 Conference, Lund, Sweden, March 10-12.

Elmqvist H., Mattsson S.E. and Otter M. (2014). Modelica extensions for multi-mode DAE systems. Modelica'2014 Conference, Lund, Sweden, March 10-12.

Lair W, Ziani R., Mercier S. and Roussignol M. (2010). Piecewise Deterministic Markov Processes and Deterministic quantification with a finite volume algorithm: a study case. Congrès de Maîtrise des Risques et de Sûreté de Fonctionnement, La Rochelle, October 05-07.

Marseguerra M., Zio E. (1995). The cell-to-cell boundary method in Monte Carlo based dynamic PSA. *Reliability Engineering and System Safety*, volume 45, pages 199-204.

Otter M., Malmheden M., Elmqvist H., Mattsson S.E., Johnsson C. (2009). A New Formalism for Modeling of Reactive and Hybrid Systems. Modelica'2009 Conference, Como, Italy, Sept. 20-22, 2009.

Tombuyses B., Aldemir T. (1996). Continuous cell-to-cell mapping and dynamic PSA. Proceedings of ICONE 4 conference, pages 431-438.

Zhang H., Dufour F., Dutuit Y. and Gonzalez, K. (2009). Piecewise deterministic Markov processes and dynamic reliability. Proceedings of the Institution of Mechanical Engineers, Part O: *Journal of Risk and Reliability*, volume 222(4), pages 545–551.

Zhang H., Saporta B., Dufour F. and Deleuze G. (2013). Dynamic reliability by using Simulink and Stateflow. *Chemical Engineering Transactions*, volume 33, pages 529-534.

KB3 workbench information and download http://rdsoft.edf.fr/

Modelica language and tools: http://modelica.org