DE LA CONCEPTION A LA SURETE DE FONCTIONNEMENT : PASSERELLE ENTRE SIMULATION PHYSIQUE ET ANALYSE DE RISQUES

FROM DESIGN TO DEPENDABILITY: A BRIDGE BETWEEN PHYSICAL SIMULATION AND RISK ANALYSIS

Marc Bouissou
EDF R&D
7 bd Gaspard Monge, 91120 Palaiseau
Marc.Bouissou@edf.fr

Lena Buffoni and Bernhard Thiele
Linköping university
SE-581 83 Linköping, Sweden
Lena.Buffoni@liu.se Bernhard.Thiele@liu.se

Résumé
Modélisation est un langage de modélisation qui a été créé afin de faciliter la description de systèmes multi-physiques grâce à une approche orientée objet. Les modèles Modélica représentent habituellement seulement le fonctionnement nominal des systèmes et sont utilisés pour les simuler à des fins de conception. Cet article propose une méthode développée dans le projet européen MODRIO pour obtenir des modèles de fiabilité à partir de ces modèles Modélica, aussi automatiquement que possible. Le principe consiste à associer à la structure du système une bibliothèque écrite en Figaro, un langage de modélisation dédié à la fiabilité. Cette association permet l'utilisation des outils de la plateforme Figaro, spécialement conçus pour l'analyse de la sûreté de fonctionnement ; il est alors possible de générer un arbre de défaillance. Le traitement de l'arbre de défaillance peut donner une évaluation quantitative de la fiabilité du système, mais aussi une liste de scénarios de test pour le modèle de simulation. Chaque coupe minimale de l'arbre de défaillance peut être utilisée comme un «stress test» pour le modèle de simulation détaillé.

Summary
Modélica is a modeling language which was created in order to ease the description of multi-domain physical systems thanks to an object oriented approach. Modélica models usually represent only the nominal functioning of systems and are used to simulate them for design purposes. This article proposes a method developed in the MODRIO European project to derive dependability models from such Modélica models, as automatically as possible. The principle is to associate to the system structure a library written in the Figaro reliability modeling language. Such an association allows the use of the Figaro workbench tools, specifically designed for dependability analysis; it is then possible to generate a fault tree. The processing of the fault tree can yield a quantitative evaluation of the system dependability, but also a list of test scenarios for the simulation model. Each minimal cut set of the fault tree can be used as a "stress test" for the detailed simulation model.

Introduction

Nowadays, the design of systems frequently relies on some kind of simulation. It would be too costly to build physical prototypes and test them to validate a new architecture. There is a large variety of simulation tools for multi-physics systems, but a significant part of these tools rely on the Modélica modeling language (Fritzson 2014). The creation of this language was a breakthrough because it is close to the mathematical description of phenomena.

The purpose of reliability, and more generally, of dependability studies is to evaluate non-functional performances, i.e. to calculate probabilities of undesirable events such as the failure of the mission of a system, or to estimate the probability distribution of quantities like: 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 designed the Figaro modeling language in 1990 (Bouissou et al. 1991). This language generalizes all the above cited models, and allows casting knowledge in 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 most other reliability analyses at EDF.

This paper shows that it is possible, in order to optimize the design of a system, to establish some links between the two kinds of studies (design and dependability studies). A practical application based on a bridge between Modélca and Figaro is described.

Designer and reliability engineer: two different jobs

At EDF (like in many other companies) there are traditionally two distinct workflows for design and for dependability analyses, split across different departments. In the first workflow, designers build Modélca (or Simulink etc.) 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. Ideally the two workflows should be in parallel, with frequent exchanges between them. But in reality, reliability studies are performed long after design, just to check that the system dependability is acceptable.

This is unfortunate for several reasons: unless the reliability analysts find a major flaw in the system, the design is not reworked because of dependability issues. This is far from an optimization process! If the design is changed, it is at a high cost because this decision arrives too late in the life cycle of the system. The fact that the information is transferred manually from design models to reliability models is costly and creates the risk of introducing discrepancies and errors.

Things are progressively changing as a result of model-based system engineering approaches. More and more companies have started using SysML as a way to federate the diverse models used all along the design and validation of systems, but some other ways can be envisaged to overcome certain limitations of this approach, as explained in the next section.
Bridging the gap between designers and reliability engineers

Depending on the maturity degree of the project of a new system, various solutions can be envisaged to create a link between design models and dependability models.

At an early stage, Model Based System Engineering (MBSE) extensively uses UML and/or SysML models in order to describe in a semi-formal way the architecture of the system and its main functions. Ideally, such an UML or SysML model later becomes the center of a daisy whose petals correspond to more specialized, formal models used for simulation of physics, worst case execution time analysis of the control parts, safety and security analysis etc.

There are numerous tools on the market to build UML and SysML models. We will only quote Papyrus, because it has the unique advantage of being free, open source and already connected to dependability analyses via its module Sophia (Yakymets et al., 2015). The opportunity of creating a connection between Papyrus and the Figaro tools, and also between Papyrus and the EDF tool called PyCATSHOO that was developed in order to model stochastic hybrid systems is being examined by EDF and CEA.

A particularly hard problem is the specification of requirements and the assurance of the traceability of these requirements towards all the various models and analyses performed all along the lifecycle of a system. Most of the time requirements are written in natural language, in extremely thick documents that can contain omissions, ambiguities, not to say errors and inconsistencies. Using SysML improves the situation, but does not eradicate completely these difficulties. In the European project Eurosylsib (Eurosylsib 2010), there was a work package dedicated to the expression of requirements and links to safety analyses. This resulted in a “Properties Library” in Modelica that could be used to complement a Modelica simulation model by the expression of requirements. This solution was not completely satisfactory as it forced to “mix” the requirements with the model parts that simulate the physics of the system. In order to tackle this problem, EDF has developed (in the framework of the European project MODRIO that followed Eurosylsib) a new domain specific language called FORM-L (Nguyen T. 2016), independent from Modelica and dedicated to specification of requirements for cyber-physical systems. This language is designed in order to be relatively easy to read and write by system designers, and to allow to formally specify envelopes of trajectories the system must stay within. EDF will develop translators from FORM-L to simulation languages like Modelica in order to automatically generate observers that will be added to Modelica simulation models; such observers will check that requirements are not violated in a large number of simulations and raise warnings if they are violated. There will also be a translator from FORM-L to Figaro, in order to facilitate early dependability analyses. The FORM-L checker and compiler is currently under development; in the meantime, it is already possible to use a “shortcut” between Modelica and Figaro. Benefits one can expect from such a bridge are: early feedback, closer interactions between the design and dependability analyses, as it will be illustrated in the present paper.

A previous paper (Bouissou & De Bossoreille 2015) explained different techniques for building a dependability model from a Modelica model. Depending on the tightness of coupling between the continuous processes and discrete events such as failures and repairs, the methods proposed in this paper 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.

In (Bouissou et al., 2014) we showed that it is possible to add stochastic behavior 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 behavior can be abstracted into a discrete behavior, 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 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.).

Here we present the second approach on a realistic use case, of medium complexity.

Modelica language and tools

Modelica is a modeling language standardized by the Modelica Association (a non-profit organization that defines the Modelica language and Modelica standard library, https://www.modelica.org/). Several Modelica simulation environments exist on the market.

The Modelica modeling language

Modelica is a non-proprietary, object-oriented, equation based language. Its main objective is to model the dynamic behavior of technical systems consisting of mechanical, electrical, thermal, hydraulic, pneumatical, control and other components. Behavior is described declaratively using mathematical equations. Object-oriented concepts are used to encapsulate behavior and facilitate reuse of model components. Modelica libraries with a large set of models are available (under free as well as commercial licenses). Particularly notable, the open source Modelica Standard Library (MSL) which contains about 920 model components and 620 functions from many domains. The simulation experts of the Modelica Association develop the Modelica language standard (the current standard version is 3.3) and the open source Modelica Standard Library since the late 1990s. A comprehensive introduction to the language is provided in (Fritzson 2014).

Modelica tools

Modelica is a language standard, not a simulation environment. Several simulation environments (tools) for Modelica are available, commercially or free of charge (open-source). Well known commercial implementations include Dymola®, SimulationX®, Wolfram SystemModeler®, MapleSim®, and LMS AMESim®; open-source implementations include OpenModelica and JModelica.org.
OpenModelica

OpenModelica is an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage. Its long-term development is supported by the Open Source Modelica Consortium (OSMC), a non-profit organization. The goal with the OpenModelica effort is to create a comprehensive open-source Modelica modeling, compilation and simulation environment based on free software. OpenModelica has become very competitive in comparison to commercially available Modelica tools in terms of supporting Modelica language features and reliable simulation of numerically challenging problems. Due to its open-source nature it is possible to adapt and extend every part of the system. This makes OpenModelica an attractive platform for research and experimental extensions and facilitates integration efforts with other tools. OMEdit (Figure 1) is a graphical front end that provides the users with easy to use model creation, connection editing, simulation of models, and plotting of results. OpenModelica has been developed since 1998 and runs on Windows, Linux, and Mac operating systems.

![OMEdit graphical Modelica editor](image)

**Figure 1. OMEdit graphical Modelica editor**

The Figaro language and tools have been developed at EDF since 1990. The language definition is open source.

**The Figaro modeling language**

The Figaro language, created in 1990, is a domain specific object oriented modeling 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 modeling 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;
- 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.

**Figaro tools**
The tool KB3 was designed to offer a generic graphical user interface for working with Figaro models; this GUI 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.

![Figaro workbench overall architecture.](image)

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 & Houdebine 2002).

**Connecting Modelica tools to Figaro tools**
The approach described in this paper relies only on the binding of a standard Modelica model to a Figaro library containing the information specific to reliability studies. This has several advantages, going far beyond the mere saving of development efforts:
- Modelica models can be left unchanged: there is no risk of introducing bugs in validated models by trying to extend them towards dependability applications;
- 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 abstractions are used when passing 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.
- Having a link between the two models ensures the coherence between them when changing one of the models and reduces the amount of information that needs to be input.

Starting from a pure Modelica model designed for physical simulation and annotated with some Figaro-specific information, one can obtain a Figaro model by extracting the objects relevant to the dependability analysis (not all the objects in the physical representation are necessarily used in the dependability analysis) 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). This process is described in detail, with examples, in the public MODRIO deliverable (Bouissou 2016). This procedure can work only because of a unique feature of the Figaro language, that is absent from all other formalisms allowing to derive dependability models from SysML, Modelica, Matlab-Simulink etc. design models: the possibility to write libraries with such a high degree of genericity that the user never has to modify the rules of the library: the user only has to assign values to string or numerical parameters (the approach is illustrated in the application example below). On the contrary, with tools like Sophia or HiP-HOPS (Papadopoulos and McDermid 1999) and its various descendants, the user has to write rules for each component, even if many of them are identical. Even with Altarica 3.0 (Batteux et al. 2015) the latest avatar of the Altarica family of modeling languages dedicated to dependability analyses, the situation is not much better. This language has a syntax very much inspired by the Modelica syntax, but in contrast with Modelica and Figaro, its structure is too rigid to enable the definition of components whose number of inputs is not known in advance. Detailed explanations about this limitation can be found in the article (Bouissou and Seguin 2006). This paper was written on the basis of Altarica dataflow that had a quite different syntax from Altarica 3.0, but several limitations of Altarica described in this paper still exist. For example, suppose that an Altarica library for electrical systems contains the description of a busbar with, say 3 connections, the user may have to add to the library the description of busbars with 2, 4, 5 etc. connections depending on the system he has to model. Besides the fact that the definition of rules for a busbar with many connections is very cumbersome and error prone (in Figaro the rules do not depend on the number of connections because they can use quantifiers), the automatic mapping of Modelica components (for which this distinction does not exist) to Altarica components would require the counting of connections, which is quite inelegant.

There are two possible levels of automation in the process of mapping a Modelica to a Figaro model.

The **first level** 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:
- as explained above, the structure of the Figaro model may be quite different from the structure of the Modelica model;
- different Figaro libraries can be bound to a single Modelica model. This will be illustrated in the next section.

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, the ways connections between components are declared are quite different in Modelica and Figaro. In order to generate Figaro 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 has not yet been implemented in the tools.

**Application: design of a reliable architecture for a power supply system**

The use case chosen to illustrate the approach and demonstrate the link between the Modelica and Figaro system models is a part of the electrical supplies of a nuclear power plant. This system uses complex reconfiguration strategies for coping with component failures and thus its reliability analysis is a difficult task. However the size of the system is such that a fairly detailed description of the system and the used transformations can be given within this paper.

**System description**

The system is the supply of 6.6 kV emergency bus bars in a 1300MW nuclear power plant. These bus bars play a crucial role in the safety of the plant, as they supply control systems and the components necessary to cool the core. Given their importance, these bus bars are fed with a lot of redundancies. In a normal situation, they divert some power from the output of the plant. In case of a loss of the 400kV line on which the plant sends its output another line takes over. If all external sources are lost, an auto-feeding of the plant can be tried. Finally, as a last resort, two diesel generators can be used. This system is both recongifurable and repairable, with the risk of failure of reconfigurations (refusal of opening or closing of circuit breakers). Diesels operate only rarely and for relatively short durations: the frequency and duration of their functioning depend on other components.

**Figure 3. Simplified representation of the 6,6kV emergency power supply of a NPP**
Modelica model

A Modelica model of the emergency power supply has been developed in order to illustrate the linkage of Modelica and Figaro systems. The model uses the PowerSystems library for electrical power systems as a base (Franke and Wiesmann, 2014). Modeling the failure of components and allowing for system reconfigurations required the development of new model components which are collected in a Modelica library named EmergencyPower. All components are based on the phase systems modeling framework of the PowerSystems library. The presented emergency power supply model is parametrized for using the three-phase system in modal (dq0) coordinates. Figure 4 shows the component diagram view of the model.

Figure 4. Modelica model of the 6.6kV emergency power supply.

Figure 5. Outputs of the simulation of a sequence of failures (time is in seconds).
Faults can be injected in the bus bars and circuit breakers. Faulty components are marked by a red flash. The simulation results of Figure 5 correspond to a possible "interpretation" of the minimal cut set: LGA001TB6K6_FR LGD001JA6K6 FO LGB001TB6K6_FR LGF002JA6K6 FC LHA001JA6K6 FO LH002JA6K6_FC, found with the help of the fault tree generated by the method described in the next sections. This cut set is mainly composed of failures on demand. The two failures in function LGA001TB6K6_FR and LGB001TB6K6_FR can happen in any sequence. In the Modelica model, the two faulty bus bars are configured to experience a failure to run (FR) after 10s (LGA001TB6K6) and 20s (LGB001TB6K6), respectively. The failure to run is modeled as a short to ground of all three phases. The faulty circuit breakers may either experience a failure to open (FO), or a failure to close (FC). If a circuit breaker is closed a fault sensing model is active that detects excess of current flowing through the device or a lack of voltage potential to the ground (the respective thresholds are parametrized). If the sensor values exceed the thresholds for a (parametrized) consecutive time duration an error is detected and signaled over a Boolean output port (pink triangle) and the circuit breaker opens. If two circuit breakers are connected to each other, the error output is used to signal to an initially open circuit breaker that it shall close. Since Modelica does not allow structural changes of the equation system during simulation the difference between an open and a closed circuit breaker is in the magnitude of the electrical resistance of the circuit breaker. The transition between an open and closed circuit breaker is modeled by a continuous change of the current-voltage ratio using an exponential relaxation function. This modeling approach showed good-natured numerical behavior during simulation.

The power output of the plant is modeled as an ideal prescribed power source (TRA003APGRP), the power grid is modeled as an ideal voltage source (GEV001RE400), ideal voltage converter components are used for transformers (GEV001TP20K, GEV001TS400, GFR001TA400), the diesel generators are modeled as ideal power sources (LHP001GE6K6, LHD001GE6K6), the critical power loads are modeled as impedance to ground (pte_tension_LHA, pte_tension_LHB). Hence, for this example many ideal component models are used. However, notice that the PowerSystems library also contains components for detailed transient modeling of three-phase power systems which can be easily substituted for the ideal components if the transient behavior of the components is essential for the analysis at hand.

Figure 5 shows a graph of the power consumed by the electrical loads during the simulation. Notice that the power drops at the failure of the LGB001TB6K6 bus bar at t=10s. The system tries to reconfigure two times, but due to the FO faults in the circuit breakers, the line is finally lost. The LGB001TB6K6 bus bar fails at t=20s, but due to the FC faults in the circuit breakers neither of the two backup configurations can establish a power link.

**Figaro model**

There are several Figaro libraries currently in use at EDF for assessing the reliability and/or availability of electrical systems. They can be classified in three categories:

- those used in the framework of Probabilistic Safety Analyses of nuclear power plants for the automatic generation of fault trees;
- those describing the dynamic behavior of electrical systems. Among them is the library K6, used for the assessment of reliability and availability of distribution systems for Internet hotels, hospitals, airports, factories... K6 is presented in another paper of lambda-mu 20 (Chaudronneret et al., 2016), which also gives a more detailed "catalog" of Figaro libraries for electrical systems;
- the library BDMP. This one is an "abstract" knowledge base: it requires more work from the analyst to build a model, but it is very flexible since it is based on a general purpose mathematical formalism (Bouissou & Bon 2003) instead of physical components.

The existence of the first two categories shows that depending on the context, it may require different simplifying assumptions to study a given system. It would be possible to link the Modelica model to any library in these two categories; with K6, it would be possible to carry out a precise, quantitative dependability analysis. But this would not fit with our purpose of finding, as efficiently as possible, minimal cut sets of the system.

Hence, the library we used for this article is a generic library called ELEC, suitable for generating fault trees for power supplies in any French nuclear power plant. The very first versions of this library were developed just after the creation of the Figaro language, in the 90's. This library is currently composed of 988 lines of Figaro language (including comments); it offers 36 kinds of components that can be assembled to create models. What justifies the use of a fault tree (a static model) for modeling a highly dynamic system is the fact that the components are considered non repairable and that the mission time is short enough to keep probabilities of failures small. In such conditions the quantification of the fault tree by the sum of probabilities of minimal cut sets is generally pessimistic, and a fairly good estimation (Bouissou 2008).

The principles for generating fault trees with KB3 from ELEC differ from the algorithm described in (Salem et al., 1976), which is still used in most, if not all non Figaro-based tools that "generate" fault trees nowadays. This simple algorithm is suitable for control systems where components have clearly identified inputs and outputs, but is not sufficient for modeling physical systems where interactions can propagate in any direction; this criticism was already formulated in (Squellati 1980), more than thirty years ago!

The minimal inputs needed by the ELEC library are the system topology, the so-called "initial state" and "mission state" of components, and what is called "leak blocks" (the term leak is used to denote the analogy between a short circuit in an electrical system and a leak in an hydraulic system). In addition, the user can modify the generated fault tree by inhibiting some failure modes of certain components, and modifying the default values for reliability data (failure rates of components).

Here are explanations about the initial and mission state, and leak blocks.

The initial state describes the active parts of the system at the start of the mission: this is determined by closed circuit breakers and by the active power sources. The mission state is generally a state in which all backup sources are active (in addition to normal sources) and all circuit breakers are closed. This is obviously far from reality, but this information is exploited to determine which failure modes must be taken into account in the fault tree for circuit breakers, and, less importantly, for some other components. For example, the "fail to close" (FC) failure mode of a circuit breaker appears in the fault tree if and only if its initial state is "open" and mission state is "closed". The "spurious opening" (SO) appears if the initial or mission state is "closed" etc.
The leak blocks are built manually with the ELEC library, but this construction could be automatic from the system topology, as in the OPALE library (Chaudonneret et al., 2016). Their role is to allow the propagation of effects of short circuits in all directions in the system, regardless of the normal direction of the flow. A leak block is a set of components that cannot be disconnected one from another by circuit breakers. Therefore, when a short circuit occurs in a leak block, all components of the leak block must be considered as lost. ELEC uses another kind of (automatically built) “functional” component, called obstruction blocks. An obstruction block is a set of components is series, between two nodes having more than two connections. Another way to define leak and obstruction blocks is to consider that:

- leak blocks are equivalence classes for the relation; has the same voltage whatever the circuit breakers positions;
- obstruction blocks are equivalence classes for the relation: is traversed by the same current intensity whatever the system state.

The interaction rules in the ELEC library can be divided in two categories: there are rules defining the propagation of voltage from the faulty/nominal state of leak and obstruction blocks, and rules defining these states as a function of the state of components.

Thanks to the existence of leak and obstruction blocks, it is possible to avoid the necessity to have clearly identified inputs and outputs for each component. When the interaction rules are transformed into a fault tree for a given system and given points of interest (like the bus bars LHA and LHB in our example), it produces a fault tree with two levels: the top of the tree contains gates related to blocks while the lower level contains gates relating the components failures to block failures (cf. Figure 7).

**Deducing the Figaro model from the Modelica model**

During (or after) the creation of the Modelica model, Figaro code is added in strings, in certain components composing the model. The “augmented” model can then be used either for the usual deterministic simulation, or for dependability studies based on a Figaro model. To obtain the Figaro 0 model, Figaro specific information must at first be extracted from the Modelica model and gathered into a single file containing all the Figaro objects. Then the Figaro processor checks the consistency of this file in itself and with regard to the Figaro knowledge base, and creates the Figaro 0 model.

At that point, all Figaro processing tools, including, but not limited to, the fault tree generator, can be used.

In order to simplify the introduction of the Figaro strings, a simple adaptation of the Modelica library is needed. Each class C of the Modelica library that will contain Figaro strings must be declared to extend one of the two classes Figaro_Object or Figaro_Object_connector, contained in a Modelica package called FIGARO:

```modelica
package FIGARO
model Figaro_Object
  "Generic structure of objects that will contain Figaro code"
  parameter String fullClassName
  "Name of the class the object will belong to in the Figaro library"
  parameter String codeInstanceFigaro
  "Figaro code specific to the current instance"
end Figaro_Object;

connector Figaro_Object_connector
  "Generic structure of connectors that will contain Figaro code"
  // the added parameters are the same as for Figaro_Object
  parameter String fullClassName
  "Name of the class the connector will belong to in the Figaro library"
  parameter String codeInstanceFigaro
  "Figaro code specific to the current instance"
end Figaro_Object_connector;
end FIGARO;
```

The code is the same in these two classes, but it is necessary to duplicate it because in Modelica a connector can only inherit from a connector. This package FIGARO must be loaded when using Figaro extensions, whatever the tool used.

Next, we modify the components in the Modelica model to include the Figaro reliability information. This is done in two steps. The first step is to map the Modelica components and connectors to their Figaro counterparts. This is done through inheriting from the Figaro library. At this level the fullClassName is specified, but can be overridden for an individual instance if necessary. The code snippet below shows the mapping between the ForcedBreaker class in Modelica and the circuit_breaker class in Figaro. In this case we only have objects and no links. It is also important to note that not all the physical components are modeled in the reliability analysis (for example LoadImpedance components are not modeled in Figaro).

```modelica
model ForcedBreaker
  "Breaker, forced change of current-voltage ratio with exponential relaxation"
  extends FIGARO.Figaro_Object (fullClassName = "circuit_breaker");
  import SI = Modelica.SIunits;
... end ForcedBreaker;
```

The second step of the mapping between Modelica and Figaro is done object by object. For example, Figure 6 shows the Figaro instance code associated with a given circuit breaker. The information on the connections between components is stated after the keyword INTERFACE. Information about the initial and mission states is stated after the keyword CONSTANT.
Generation and exploitation of a fault tree

During the MODRIO project, two prototypes were developed, in the tools Dymola and OpenModelica. With any of them, it is possible to load the Modelica model including the Figaro indications and then run a command that generates the fault tree. What happens behind the scene is the operations described in the previous section, plus a second call to the Figaro processor in order to generate the fault tree, in the form of an XML file. Figure 7 shows two extracts of the fault tree, taken respectively in the upper and lower levels of the tree. The whole fault tree is made of 45 gates and 51 basic events. It is worth mentioning here that the exact same fault tree would have been built by a reliability specialist using the KB3 platform with the ELEC library; but this would have required a completely manual input of the information already available in the Modelica model. Once the fault tree is available, it can be loaded in a standard tool for obtaining minimal cut sets with their probabilities, calculating the top event probability etc. This is a standard reliability analysis. The results of this study can be exploited in two ways: on the one hand, to check that the system reliability is sufficient, and if not, suggest changes in the system architecture. On the other hand, the minimal cut sets can be used, in an exhaustive way or by sampling (this looks more reasonable, as this fault tree has 29353 minimal cut sets with orders from 2 to 6), as explained above to check the correct functioning of reconfiguration procedures induced by the control of the system. If the detailed Modelica simulation shows that after n failures taken in a minimal cut set of order p, the system fails, this shows that the control does not work properly and leads to premature failures. Using the minimal cut sets as “stress test” scenarios is far more efficient in terms of coverage than generating failures at random.

Conclusion

In the context of 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 through an automatic transfer of information towards mature dependability tools based on the Figaro modeling language.

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.

There are other possible uses of this link between Modelica and Figaro. In fact all existing uses of Figaro models can also be exploited: interactive simulation, Monte Carlo simulation, search and quantification of sequences leading to a failure state, etc.
Figure 7. Two excerpts of the generated fault tree

References


Bouissou M. (2016), Specification of Modelica extensions and interfaces for Bayesian networks and Fault trees. Deliverable D.2.2.1 of the MODRIO Artemis project.


Eurosyslib ITEA2 project: www.eurosyslib.com


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

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

MODRIO ITEA3 project: https://itea3.org/project/modrio.html


OpenModelica, Open Source Modelica Consortium (OSMC): http://opencmodelica.org/


