Hindawi
Modelling and Simulation in Engineering
Volume 2017, Article ID 1578043, 15 pages
https://doi.org/10.1155/2017/1578043

Research Article

Model-Based Dependability Analysis of Physical Systems with Modelica

Andrea Tundis,[1] Lena Buffoni,[2] Peter Fritzson,[2] and Alfredo Garro[1]

[1]*Department of Informatics, Modeling, Electronics, and System Engineering (DIMES), University of Calabria, Via P. Bucci 41C, 87036 Rende, Italy*
[2]*Department of Computer and Information Science (IDA), Linköping University, 581 83 Linköping, Sweden*

Correspondence should be addressed to Andrea Tundis; andrea.tundis@dimes.unical.it

Received 15 August 2016; Accepted 27 December 2016; Published 8 March 2017

Academic Editor: Franco Ramírez

Modelica is an innovative, equation-based, and acausal language that allows modeling complex physical systems, which are made of mechanical, electrical, and electrotechnical components, and evaluates their design through simulation techniques. Unfortunately, the increasing complexity and accuracy of such physical systems require new, more powerful, and flexible tools and techniques for evaluating important system properties and, in particular, the dependability ones such as reliability, safety, and maintainability. In this context, the paper describes some extensions of the Modelica language to support the modeling of system requirements and their relationships. Such extensions enable the requirement verification analysis through native constructs in the Modelica language. Furthermore, they allow exporting a Modelica-based system design as a Bayesian Network in order to analyze its dependability by employing a probabilistic approach. The proposal is exemplified through a case study concerning the dependability analysis of a Tank System.

1. Introduction

System dependability [1] represents an important requirement to be satisfied for a wide range of systems. It is even crucial when a system failure may lead to catastrophic losses in terms of cost, environmental damage, or human lives, as in several industries such as in automotive, energy, and aerospace industries [2–5].

Typically, the dependability analysis is carried out through different approaches, some of them based on the use of classic qualitative or quantitative techniques [6]. In particular, qualitative analysis techniques aim to identify the possible system failures, their rate of occurrence, and (local and global) effects on the system so as to individuate corrective actions and make them more fault tolerant [7, 8]. Two main techniques are mostly exploited: FMECA (Failure Modes Effects and Critical Analysis) and FTA (Fault Tree Analysis) [9–11], which are focused on the structural analysis of the system. Quantitative analysis techniques (such as Series-Parallel system reliability analysis and Markov Chains) [12–14] are based on the identification and modeling of physical and logical connections among system components and on the analysis of their reliability through statistical methods and techniques. In addition, for taking into account the increasing importance of software components in modern systems, most of the above-mentioned techniques have been extended for embedded and software intensive systems, for example, S-FMECA (Software Failure Modes, Effects, and Criticality Analysis) and Software Fault Tree Analysis (S-FTA) [15]; moreover, specific software-oriented techniques have been also introduced such as the following: (i) HSIA (Hardware-Software Interaction Analysis), which is used for analyzing that the software is specified and designed to react to hardware failures and to ensure that it cannot overstress hardware; (ii) SCCFA (Software Common Cause Failure Analysis), which is intended to determine and define corrective measures for potential software failures in multiple systems or multiple subsystems; and (iii) PSH (Product Service History), which is an approach that can be followed to demonstrate software maturity [16]. In more dynamic

contexts, model-based and simulation-driven approaches [17–21] are emerging to perform system monitoring and verification against requirements and dependability analysis by analyzing the evolution of the system and its emergent behavior.

Unfortunately, the increasing complexity of systems and the accuracy required in their dependability analysis make, often, inadequate such techniques and the verification and fault analysis processes even harder to sustain. Moreover, the application of these techniques, though applied to the same system, requires the redefinition and an ad hoc implementation of the same system model for each analysis technique to be applied. This implies waste of time and costs in terms of money, with the possibility of having errors during each stage of the system development/modeling process; as a consequence, only some of such techniques are adopted or partially applied to subset of system components.

The demand of analyzing the dependability performances of a system on different simulation tools is increasing [22]. In this context, a common system representation could bring many advantages, specifically the following: (i) reusability of system models; (ii) integration of different system models and their (co)simulation on heterogeneous and distributed environments; (iii) combination and assessment of results gathered from the simulation of the same system model on different platforms. In fact, the comparative analysis of the results could be useful to validate and evaluate the representativeness of the model, whereas the combination of the results could allow exploiting specific and complementary features of the considered platforms and thus enabling a more comprehensive study of the system and its different aspects. In fact, it would be important to be able to (i) integrate, within the simulation analysis model, requirements or properties to be checked and be tested in advance; this would allow studying the behavioral aspect in the design, before realizing the system and obtain an assessment in advance; (ii) use the same model in different tools in order to perform other types of analysis, comparative or complementary, without having to redesign and reimplement it. As a consequence, there is a strong demand for new, more powerful, and flexible models, analysis tools, and techniques which are also centered on model-based approaches that benefit from the available modeling practices and incorporate the use of simulation to flexibly assess the dependability indices of systems and evaluate design choices according to different perspectives.

Specifically, Modelica-based solutions represent valuable candidates due to their popularity especially in industrial domains, centered on complex physical systems, in which dependability is a fundamental property to be evaluated; however, to enable these features it is necessary to face some main challenges. In particular, there is a need to (i) introduce additional concepts and, as a consequence, abstraction levels in the model for representing such additional system characteristics (e.g., properties, requirements) and (ii) provide (semi)automated mechanisms and rules for supporting the transformation of models and make them available for being analyzed on other external tools.

Starting from this consideration and the above-mentioned challenges and needs, the paper proposes a solution mainly centered on the Modelica language [23] for supporting dependability analysis of systems centered on stochastic Bayesian Network (BN) (also called *belief networks*, *Bayesian belief networks*, *causal probabilistic networks*, or *causal networks*) [24]. BNs are a useful tool to evaluate the dependability of systems, in which nodes represent variables and arcs represent direct probabilistic dependencies among them. Specifically, BNs allow specifying dependency models that represent the key factors and their interrelationships (a qualitative model) with probability distributions based on expert judgment or from observed data (a quantitative model) [25]. To this end, a set of new constructs to the Modelica language are introduced, as well as a methodological process for integrating requirements into the development process. This approach allows benefiting from the reuse of models by enriching the Modelica languages through specific extensions that enable not only the evaluation of the behavior of system during its evolution but also its structural assessment. Moreover, based on such constructs, an algorithm has been implemented for enabling the automatic generating of BN, ready to be analyzed through another external tool called GeNiE [26], which is able to perform the dependability analysis based on a probabilistic model.

The rest of the paper is organized as follows: Section 2 provides an overview on the Modelica language and related simulation environment; Section 3 introduces the Bayesian Network and discusses the importance of reusing models; the proposal in terms of Modelica extensions and their implementation in OpenModelica are described in Section 4, whereas an example on how to use such extensions for performing dependability analysis is shown in Section 5; related works are discussed in Section 6 whereas conclusions are drawn in Section 7.

2. The MODRIO Project and Motivations and Role of Modelica

The results presented in this paper have been developed in the context of the ITEA 3 MODRIO (MOdel-DRIven physical systems Operation) European Project [27] that aims to support model-driven physical system operation, an important branch of systems engineering. The wide interest in this research field is constantly increasing as evidenced by the presence of 38 important European partners involved in the project among industrial and academic ones (EDF, DLR, Dassault Aviation, Dassault Systèmes, EADS, Siemens, Scania, ABB, SKF, University of Calabria, Linköping University, Katholieke Universiteit Leuven, etc.).

MODRIO is devoted to addressing issues for supporting Systems Specification, Modeling and Analysis, Scalability, Complexity Management, and Validation and Verification from system design to system operation, by extending modeling and simulation languages and tools, based on open standards. In particular, the focus is on modeling and simulation techniques, models, methods, languages and tools able to cover several aspects such as conceptual properties representation, binding and automated model composition,

and tracing and verification. The expected outcome of the project is a holistic modeling and simulation framework for physical system design, diagnosis, and operation assistance.

In this context, the Modelica language and related tools, which are described in the following subsections, have been chosen for being the core tools of the MODRIO research project, thanks to their hybrid characteristics, simplicity, and flexibility in supporting the modeling and simulation of physical systems and to the possibility of extending them in order to support system dependability analysis.

2.1. The Modelica Language. Modelica is a popular equation-based modeling language for representing physical systems (e.g., systems containing mechanical, electrical, electronic, hydraulic, thermal, control, and electric power components) with *acausal* features, which allows defining models in a declarative style [23].

Modelica is a primary modeling language that allows the specification of mathematical models of complex and dynamic systems for enabling their simulation. Modelica is also an object-oriented equation-based programming language for implementing computational applications with high complexity and that require high computing performances. The main features of Modelica are the following [28]:

(i) Modelica is primarily based on equations instead of assignment statements. This allows for an *acausal* modeling approach that gives a better (re)use of classes since equations do not specify a data flow direction and thus can represent different behavioral models on the basis of the actual inputs provided during the model execution.

(ii) Modelica provides multidomain modeling capabilities, meaning that models of components corresponding to physical objects which belong to different domains (such as electrical, mechanical, hydraulic, and control applications) can be described and connected in a unified model.

(iii) Modelica is an object-oriented language with a general class concept that unifies classes, generics (called templates in C++), and subtypes into a single language construct. This facilitates both the reuse of components and the evolution of models.

(iv) Modelica has a strong software component model with specific constructs for creating and connecting components; this feature makes it suitable for the architectural description of complex physical systems as well as of software systems.

2.2. Modelica-Based Simulation Environments. Several programming environments are currently available for the design and simulation of Modelica models. Some simulation environments also have facilities for the analysis of simulation results, 3D visualization, CAD integration, specific support for real-time and hardware-in-the-loop simulation, and support for integrated modeling, simulation, postprocessing, and documentation. Basically, most of the Modelica-based environments have a number of common characteristics: (i) *a Modelica-based compiler or interpreter*, for translating Modelica code to some interpreted code form such as byte code or abstract syntax tree or to a compiled language (such as C, C++), which is further translated into executable machine code by standard compiler; (ii) *an execution and run-time module*, which makes the executable code coming from translated Modelica expressions, functions, and equations runnable; (iii) *an editor for textual model editing*, which can be any standard text editor or a special-purpose built-in text editor; (iv) a graphical model editor, which is a graphical connection editor for component based model design and allows visually connecting instances of Modelica classes.

In the following, some of the most popular Modelica-based simulation environments are briefly described:

(i) *OpenModelica*, the widest adopted Open Source Modelica-based environment that, in addition to the above-mentioned components, consists of an *interactive session handler* that parses and interprets commands and Modelica expressions. The session handler also provides simple history facilities and completion of file names and identifiers. It is based on the *RealLine* library which is available in many Linux and Unix distributions. Its long-term development is supported by a nonprofit organization—the Open Source Modelica Consortium (OSMC) [29].

(ii) *Dymola*, a comprehensive commercial modeling and simulation tool, with particular emphasis on efficient real-time simulation whose main applications are in the automotive domain, such as hardware-in-the-loop simulation of automatic gearboxes. The architecture of Dymola is similar to OpenModelica; however it contains a *symbolic optimizer* to reduce the size of equation systems as well as a *simulation run-time system* including a numeric solver for hybrid DAE equation systems. Furthermore, Dymola contains additional facilities including 3D graphic animation option, integration with Concurrent Versions System (CVS) for supporting version handling [30].

(iii) *MathModelica*, an integrated development environment which integrates Modelica-based modeling and simulation capabilities with a visual design tool, advanced scripting facilities, integration of programme code, test cases, mathematical type setting, and symbolic formula manipulation (provided via Mathematica). The user interface consists of (i) a graphical model editor in which models can be assembled using components from a number of standard Modelica libraries and (ii) an electronic notebook that is an interactive document that combines simulation models and technical computations with text, table, code, and other programming elements. The provided MathModelica internal format allows the user to extend the environment with new functionality as well as to perform queries on the models and write scripts for automatic model generation [31].

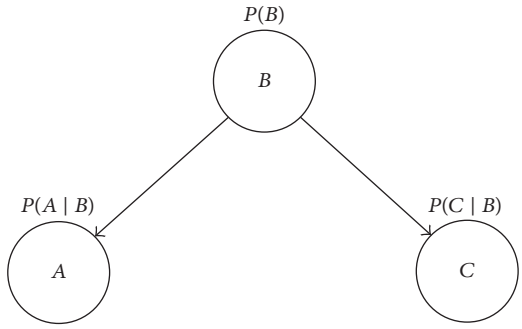

Figure 1: Example of Bayesian Network.

3. Generating Bayesian Networks

The need of enabling different kind of analysis, such as complementary or comparative, functional or performance, starting from the same system model, is nowadays increasing. As an example, dependability, which is the collective term used to describe the availability performance of a system and its influencing factors, is such an important property. Especially in mission-critical domains, dependability has to be carefully considered and evaluated when a system is being built. Indeed, a complementary analysis for evaluating faults of systems, their failures, and their propagation among system components would help to improve the system development lifecycle since its early stage of design and development. In this context a viable way to assess dependability concerns the employment of Bayesian Networks for model-based fault evaluation [25].

3.1. Background. A Bayesian Network or Belief Network (BN) [24] is a probabilistic graphical model that represents a set of random variables and their conditional dependencies through a directed acyclic graph (DAG). For example, a Bayesian Network could represent the probabilistic relationships between faults/causes and failures/consequences. Given failures/consequences, the network can be used to compute the probabilities of the presence of various faults/causes. Formally, Bayesian Networks are DAGs whose nodes represent random variables in the Bayesian sense: they may be observable quantities, latent variables, unknown parameters, or hypotheses. Edges represent conditional dependencies; nodes that are not connected represent variables that are conditionally independent of each other. Each node is associated with a probability function that takes, as input, a particular set of values for the node's parent variables and gives (as output) the probability (or probability distribution) of the variable represented by the node. For example, if m parent nodes represent m Boolean variables then the probability function could be represented by a table of 2^m entries, one entry for each of 2^m possible combinations of its parents being true or false. Similar ideas may be applied to undirected, and possibly cyclic, graphs; such are called Markov networks. As shown in Figure 1, an edge from B to A indicates that B causes A.

Conditional probability distribution (CPD) is specified at each node that has parents, while prior probability is specified at node that has no parents (the root node). As shown in Figure 1, the CPDs of variables A and C are $P(A \mid B)$ and $P(C \mid B)$, respectively; and the prior probability of B is $P(B)$. The edges in the Bayesian Network represent the joint probability distribution of the connected variables. For example, the joint probability distribution for the edge (B, A) is $P(A, B)$ which represents the probability of joint event $A \wedge B$. The fundamental rule of probability calculus shows that

$$P(A, B) = P(A \mid B) \cdot P(B) \tag{1}$$

and, in general, the joint probability distribution for any Bayesian Network, given nodes $X = X_1, \ldots, X_n$, is

$$P(X) \prod_{i=1}^{n} P\left(X_i \mid \text{parents}\left(X_i\right)\right), \tag{2}$$

where parents(X_i) is the parent set of node X_i. Equation (2) is known as the chain rule, which indicates the joint probability distribution of all variables in the Bayesian Network as the product of the probabilities of each variable given its parents values. Inference in the Bayesian Network is the task of computing the probability of each variable when other variables' values are known.

3.2. Importance and Motivations of System Simulation Model Reuse. A system can be considered as an object or a collection of objects belonging to the real world that we are interested to study in terms of its properties. On a system experimentations can be performed, where an experiment is the process of extracting information from a system by exercising its inputs. A model is a possible and simplified representation of a system on which experiment can be applied in order to answer questions about the system. As a consequence, a simulation is an experiment performed on a model.

The reuse of (simulation) models can be adopted at various abstraction levels starting from the reuse of small portions of code, through components reuse, or even fully reusing complete models. The model reuse approach brings from one side benefits and from the other side drawbacks according to the abstraction level under consideration (to be reused) [32]; in particular,

(i) concerning benefits, full models can be directly both reused as a template to better support, in a more efficiently way, similar projects on the same simulation environment and exploited by external tools for enabling further complementary analyses, whereas the reuse at component level can be exploited, for instance, to create libraries thus reducing both time and cost when compared with a new development as well as to improve the quality of simulation models. Moreover, the reusability of system models (i) could be an easier integration of different system models and their (co)simulation on heterogeneous and distributed environments and (ii) could allow the combination and assessment of results gathered from the simulation of the same system model on different platforms. Furthermore, the comparative

analysis of the results could be useful to validate and evaluate the representativeness of the model; whereas the combination of the results could allow exploiting specific and complementary features of the considered platforms and thus enabling a more comprehensive study of the system and its different aspects;

(ii) concerning drawbacks, the main issues to be faced with the reusing of models typically concern the following: (i) identification of potentially reusable components; (ii) incompatibilities among model components in the granularity level of models because of difference in the fundamental approaches of coarse-grained modeling and fine-grained modeling; (iii) representation of the objectives, assumptions, and constraint; (iv) level of flexibility in updating/modifying an existing component model; (v) interoperability of the reused components.

Intuitively, moving from the reuse of complete models towards the reuse of (sub/partial) components, the frequency of reuse increases whereas the level of complexity of the reuse decreases and vice versa.

Furthermore, when there is the need to define solutions into larger industrial contexts, additional aspects not only related to the reusability but also related to the interoperability of models, processes, and methodologies require to be faced. As an example, great attention has been specifically devoted to the reusability of Modelica models in [33]. In particular, the authors propose the design and development of a solution to store and reuse physical system models by indexing and retrieving their content and metadata. More specifically, in the Modelica context, the authors define a mapping between the representation modeling language and a semantic-based representation model.

Concerning the interoperability among models, an interesting effort is represented by the FMI (Functional Mock-Up Interface) standard [34]. FMI represents one of the most recent solutions for enabling model exchange and cosimulation and it is already supported by several Modelica-based tools such as OpenModelica and Dymola.

In this paper, the reuse of Modelica-based models, which are typically employed for performing system behavioral analysis through simulations, is exploited for supporting complementary dependability analysis. In particular, the Modelica constructs are properly transformed and where needed enriched or distilled for supporting the generation of Bayesian Network, so as to enable functional safety analysis, according to the proposed algorithm that is described in the next subsection.

4. Meta-Modelica Constructs and a Proposed Algorithm

The Modelica specifications and modeling language were originally developed as an object-oriented declarative equation-based specification formalism for mathematical modeling of complex systems, in particular physical systems. However, it turns out that, with some minor extensions, the Modelica language is well suited for another modeling task, namely, modeling of the semantics, that is, the meaning, of programming language constructs. Since modeling of programming languages is often known as metamodeling, the name Meta-Modelica programming [35] is used for slightly extending the Modelica language.

In this case, in order to enable the generation of Bayesian Networks, two fundamental constructs are natively introduced at the compiler level, the *fulfill* and the *requirement* concepts:

(i) *requirement*: which allows expressing requirements and, generally speaking, system properties as well as validating the behavior of a specific physical component model which is related to such requirement or validating interactions among different physical component models.

(ii) *fulfill*: which expresses the entailment relationship between physical component models and a requirement, as well as among requirements. Moreover, it provides the propagation process of an assessment among requirements and physical component models.

Starting from these proposed extensions an algorithm for the generation of Bayesian Networks has been defined and then specific APIs have been implemented for making such extensions and their related capability to user level available. The prototype is freely available and downloadable [36].

4.1. Concepts Definition, Grammar, and Related Semantics. Let C be a set of *PhysicalComponentModels* $\{C_1, C_2, \ldots, C_j, \ldots, C_m\}$ with $1 \leq j \leq m$, expressed through the Modelica standard language, which is employed for building the *Physical System Model*, one possible representation of the physical system being modeled under consideration,

Let A be a set of *RequirementModels* $\{A_1, A_2, \ldots, A_i, \ldots, A_n\}$ with $1 \leq i \leq n$, expressed through the proposed Modelica language extensions, which is used to verify C.

Let C' be a subset of C and A' a subset of A, such that $\text{card}(C') > 1$ and $\text{card}(A') > 1$, $A_i \notin A'$ and $C_j \notin C'$.

Let F be a set of *fulfill* expressions $\{F_1, F_2, \ldots, F_h, \ldots, F_q\}$, with $1 \leq h \leq q$, which allows defining the entailment relationship between *PhysicalComponentModels* in C and *RequirementModels* in A, as well as among *RequirementModels* in A. Let F' be a subset of F and $\text{card}(F') > 1$.

The grammar for defining a generic F_h expression is the following:

$$\begin{aligned} F_h &\coloneqq \langle \text{Body} \rangle \text{ fulfill } \langle \text{Head} \rangle ; \\ \text{Head} &\coloneqq A_i; \\ \text{Body} &\coloneqq \left\langle \left(C'\right)^+ \mid \left(A'\right)^+ \right\rangle ; \quad \text{with } A_i \notin A'. \end{aligned} \tag{3}$$

Moreover, a Probability Model (PM), which allows for a static dependability analysis of a Modelica-based system design by using probabilities, is adopted. In particular, the PM consists of a list of possible states in which a component can be (i.e.,

operating, operating in degraded mode, and broken) and, eventually, of a set of characteristics associated with each state. Such a model can allow the following: identifying the cause(s) of a failure, identifying the weaknesses in a system design in terms of culprit component, assessing a proposed design for its dependability or safety, assessing the effects of human errors, prioritizing the contributors to failure, making effective upgrades to a system, and quantifying the failure probability and contributors. In this case, each model defines two possible states: (i) *success*: which represents the probability of success of a component in performing its work and (ii) *failure*: which represents the probability of failure of a component in performing its work.

4.2. Transformation Rules from Modelica to a Bayesian Network. Starting from the preliminary definition described in Sections 4 and 4.1, four main types of *fulfill* expression are identified and, as a consequence, a set of *four transformation rules* (TR) $\{\mathrm{tr}_1, \mathrm{tr}_2, \mathrm{tr}_3, \mathrm{tr}_4\}$ are defined. Three rules are exploited for generating the BN structure of the system design model, the fourth rule is employed for matching to create the matching with the Probability Model. In particular,

(i) tr1: given a *fulfill* expression F_h if the Head A_i is not contained into any Body of other fulfill expressions, then the Head A_i is a root for the BN, and the A_i has only incoming edges;

(ii) tr2: given *fulfill* expression F_h, if the Body of a relationship contains only C', then all C' components will be leaves in the BN, only with outgoing edges;

(iii) tr3: given a *fulfill* expression F_h, if its Head A_i is contained into the Body of other fulfill relationships, then such Head A_i is neither a leaf node nor a root node; that is to say, it is an intermediate node in the BN, there will be as many incoming edges as the unique C' components of all the *fulfill* expressions having A_i as Head of such *fulfill* expressions, whereas it will have as many outgoing edges as there are the rules in which it appears uniquely in *fulfill* expression towards the same Head.

The adoption of a Probability Model (PM) allows for a static dependability analysis of a Modelica-based system design by using probabilities. A PM consists of a set of Probability Component Models (PRCM), which in turn consists of a list of possible states (e.g., operating, operating in degraded mode, and broken) and, eventually, of a set of characteristics associated with each state. Each component can work in some of such states. In order to avoid the definition of this model for every individual component, it is defined separately and then referenced by every component that needs to use it; such a model allows the following: identifying the cause(s) of a failure, identifying the weaknesses in a system design, assessing a proposed design for its reliability or safety, assessing the effects of human errors, prioritizing the contributors to failure, making effective upgrades to a system, and quantifying the failure probability and contributors.

Starting from a set of PCMs (Physical Component Models) enriched with a set of RMs (Requirements Models), where models of components and requirements are defined as a sort of component library, a PRCM can be associated with each of them with an Initialization Probability (IP) default value (see Figure 2). This means that all the instances, derived from models with the same IP, will have the same probability distribution and, as a consequence, the same initial values. So a fourth transformation rule is based on such considerations and it is defined as follows:

(i) tr4: for each node of the BN generated from a model $A_i \in A'$ or from a model $C_j \in C'$, specific probability values have to be associated. The Probability Model (PM) for the generation of the possible values number is defined as follows: $\mathrm{card}(S)^{p+1}$, where S represents the set of states of the node into the Bayesian Network and $\mathrm{card}(S)$ identify the number of states; that is, if $S = \{s_1, s_2, \ldots, s_d\}$ then the number of statuses is $\mathrm{card}(S) = d$, whereas p represents the number of the incoming edges, that is, the children of the node.

By combining these four transformation rules along with the concepts identified in the previous subsection an ad hoc algorithm has been defined and then implemented at compiler level by using the Meta-Modelica language. The pseudocode is described in Algorithm 1.

4.3. Pseudocode in Meta-Modelica Language. Here reported is the pseudocode of the proposed algorithm (see Algorithm 1) implemented in Meta-Modelica. In particular, an algorithm for the topological sort of the nodes is defined to generate ordered components by employing a top-down approach, whereas a bottom-up approach is adopted during the BN generation process, so first the leaf nodes are created and then the intermediate ones, up to the root.

Specifically, row (#1) represents the initialization step that checks the presence of fulfill rules into the Modelica standard source code. If no fulfill rules are available no BNs are generated, as described by rows (#2). If fulfill rules have been introduced into the Modelica source code, then, such rules are preprocessed, through step (#3), so as to distinguish among physical components and requirements, the root elements. Once the roots elements are identified, as specified through the step of rows (#4), multiple BN(s) are built by generating a specific Probability Model and integrating it. Finally, the termination step of row (#5) builds and makes the BN(s) available for being analyzed by external tools.

5. From a Modelica-Based Model to a Bayesian Network: A Case Study

This section exemplifies how useful and easy is the exploitation of such extensions and related tool features. In particular, a case study, concerning a Tank System, is first described and then used to show how to exploit the proposed extensions.

5.1. System Description. The reference Tank System is composed of four main physical components: Source component, a Tank component, a LevelController component, and a Sink

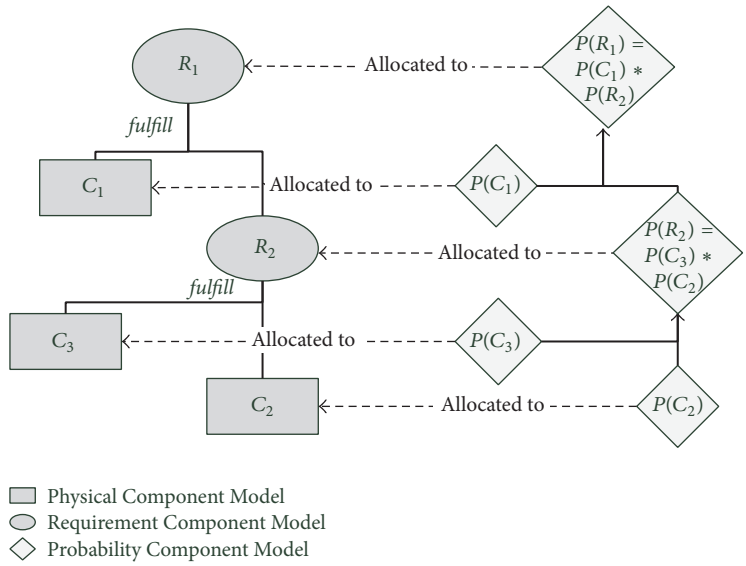

Figure 2: Probability Model integration in a Modelica-based design.

(#1) Check if *fulfill* relationships exist;
(#2) If list of *fulfill* relationships is empty then
  (a) Bayesian Network List (BNL) is empty;
  (b) Return null;
(#3) else if list of *fulfill* relationships is not empty then
  (a) Generate all (a list) the *Head* elements;
  (b) Generate all (a list) the *Body elements;*
  (c) Extract all the *root* elements from the *Head* elements;
(#4) For each *root* r element
  (a) Initialize a Bayesian Network (BN);
  (b) Generate a list L of its node-children by exploring them according to a breadth-first visit;
  (c) Compute the Probability *Model* (r, L) pm;
  (d) Map the Probability Model pm to the r element;
  (e) Extract the set C of Physical Components from the list of Children L;
    (i) For each $C_j \in C$
    (ii) Build the leaf elements of the BN (C_j);
    (iii) Compute the Probability Model (C_j) pm_j;
    (iv) Map/Integrate the Probability Model pm_j to the C_i element;
  (f) Extract the set A di Requirements/Children from the list of Children L;
    (i) For each $A_i \in A$
    (ii) Generate a list L' of its node-children
    (iii) Build the non-leaf elements of the BN (A_i);
    (iv) Compute the Probability Model (A_i, L') pm_i;
    (v) Map/Integrate the Probability Model pm_i to the A_i element;
  (g) Build BN (r, A', C');
  (h) Add the BN to the Bayesian Network List (BNL);
(#5) Return BNL;

Algorithm 1: Pseudo-code for generating Bayesian Networks from Modelica-based models.

component (see Figure 3). The Source component produces a flow of liquid, which is taken in input by the Tank component. The Tank, which is managed through the LevelController component, provides in output a liquid flow according to the control law defined by the LevelController. The Sink is the component where a part of liquid is sent.

The source code of its representation is reported in the following by using the Modelica language:

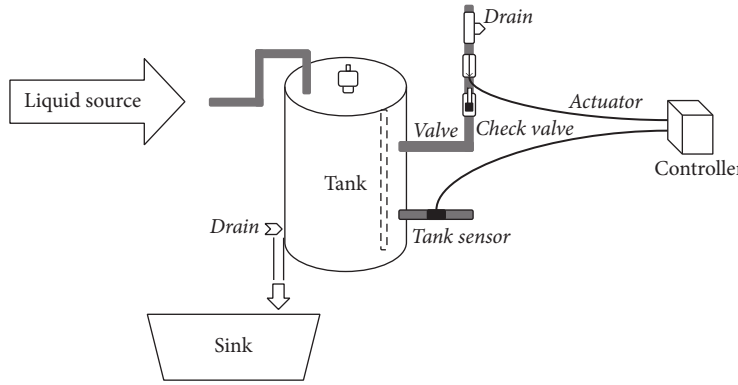

Figure 3: The reference Tank System.

```
model SystemDesign

    import TankComponents.*;
    Tank tank1(area = 1);
    Sink sink1;
    LevelController levelController
     (ref = 0.25);
    LiquidSource source (flowLevel =
     0.02);

equation

    connect(source.qOut,tank1.qIn);
    connect(tank1.tSensor,
    levelController.cIn);
    connect(tank1.tActuator,
    levelController.cOut);
    connect(tank1.qOut,sink1.qIn);

end SystemDesign;
```

5.2. Enriching the Tank System Model. Starting from the description of the Tank System, some mandatory properties (or requirements) can be defined as follows: (i) *Requirement_1* (Property_1): the liquid flow produced by the Source component should be less than 10 m^3/s; (ii) *Requirement_2* (Property_2): the role of the LevelController should be verified by exploiting both *h* level from the Tank component and the qOut flow; moreover, to be valid both *Requirement_3* and *Requirement_4* must be fulfilled; (iii) *Requirement_3* (Property_3): it takes in input both the tActuator (outflow) and the outFlowArea and checks if the outFlowArea value is proportional at the tActuator signal; (iv) *Requirement_4* (Property_4): both *h* level value and the tSensor value have to be equal; (v) *Requirement_5* (Property_5): it takes in input *h* level coming from the Tank component and checks if *h* Level < MaxLevel (in this case MaxLevel = 9) and *h* Level > MinLevel (in this case MinLevel = 5); moreover, to be valid both *Requirement_1* and *Requirement_2* must be fulfilled.

It could be misleading to express requirements by using natural languages; then, a machine oriented way would be useful to provide a more formal representation of requirements as well as to automatically verify them. As an example, *Requirements_2* can be elicited as ControlOutFlow, which takes in input *h* level from the Tank component and the qOut flow to validate the role of the *LevelController*; moreover, to be valid it requires that the Requirement_3 and the *Requirement_4* are both fulfilled. The following code shows an example of using *requirement*, in order to formalize the *Requirement_2*, described above:

```
requirement ControlOutFlow

    //qOut from the Tank component
    input Real outFlow;
    //h Level from the Tank component
    input Real levelL;
    Real proportionalCoefficient (start
    = 0);
    parameter Real threshold = 12;

equation

    (outFlow > 0) or (outFlow < 0) =
     true;
    (levelL/outFlow) > threshold = true;

end ControlOutFlow;
```

Then, it is necessary to bind the properties formalized by using the Modelica language, with the structure and the behavior of the system, under consideration. To this aim the *fulfill* construct is exploited in order to establish relationships among system components and formalized properties as well as among their values. The following code shows an example of using *fulfill*, in order to bind the component models with the requirement model:

```
{LiquidSource} fulfill limitInFlow;
{tank1,levelController} fulfill
actuateOutFlow;
{tank1} fulfill senseLevel;
```

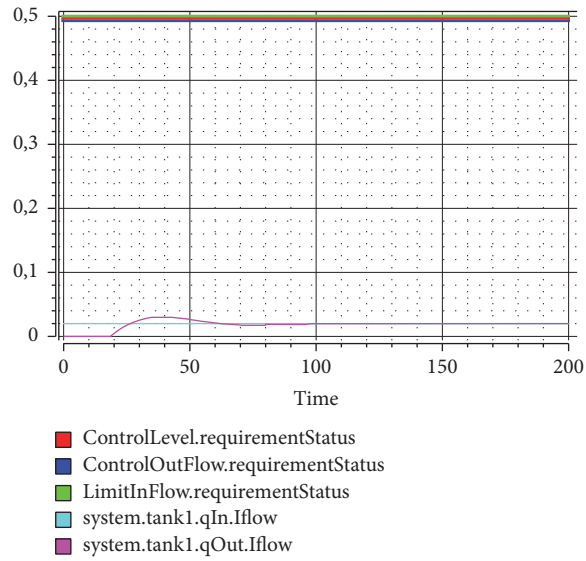

Figure 4: Red, blue, and green lines show the evaluation of requirements that are not violated throughout the execution.

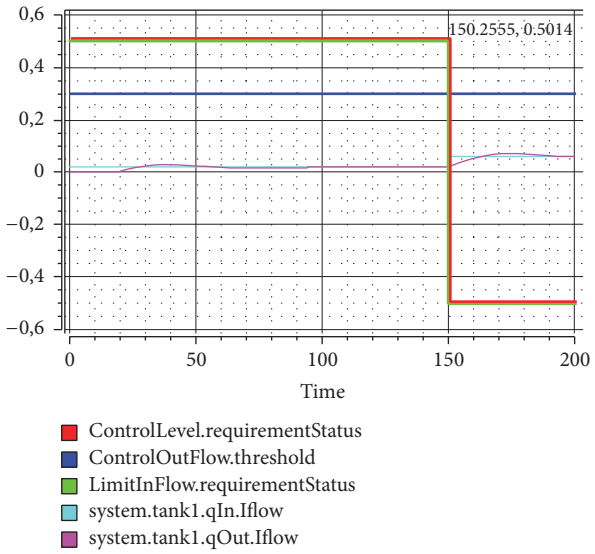

Figure 5: After 150 seconds, the status of ControlLevel and LimitInFlow requirements changes to violated.

```
{limitInFlow, controlOutFlow} fulfill
controlLevel;

{tank1, levelController, actuateOutFlow,
senseLevel} fulfill controlOutFlow;
```

By running the simulation, with a configuration that respects the constraints defined by *ControlOutFlow*, and other requirements it is possible to see that all the requirements are valid throughout the simulation (Figure 4).

Meanwhile by modifying the flow level from the source triple after 150 seconds (Figure 5), it is possible to see that the system reports a violation of the requirement *LimitInFlow*, which in turn means that *ControlLevel* is also violated.

Figure 6 shows the representation of the Tank System enriched by requirements. Starting from this initial design model and by using the Open Modelica APIs implemented, it is possible to generate Bayesian Network automatically. A first version of such approach, along with its implemented prototype, has been experienced by the authors and described in [37], where the use of the annotation mechanisms available in Modelica, to map manually the Probability Model to System Components, has been considered. Then, by experimenting it on large industrial case studies, which involve hundreds of components and thousands of relationships, such manual annotation mechanism has revealed being verbose from the developer users side. As a consequence, a further effort/improvement has been performed to make both the generation of the Probability Model and its mapping with the systems components fully automated, so as to skip/avoid the Modelica annotation mechanism.

In this latest upgraded version [36], the complexity of introduce and manage probabilities and their related values, as described in Section 4, is completely transparent to the user, and the values associated with the status of each Probability Model have the same initial value (0.50) as default, according to the PM proposed that is based on states of the following: (i) success, which represents the probability of success of a component performing its work, and (ii) failure, which represents the probability of failure of a component performing its work. The output format of the Bayesian Network is fully compliant with GeNie [26], a development environment for decision-theoretic models, such as Bayesian Networks and influence diagrams, directly amenable to inclusion in intelligent systems, developed at the Decision Systems Laboratory at University of Pittsburgh. Moreover their probability values can be freely updated by using the graphical editor of the GeNie tool [26] as well as to perform the analysis of the generated model. The source code of the generated BN is shown in Box 1. The graphical perspective is shown in Figure 7, where the circles represent the properties and the rectangles represent the components of the physical system.

Starting from the probability values associated with the Tank System model, this complementary analysis of the system with the top level represented by the requirement *ControlLevel* shows that the probability of failure is equal to 14%, whereas the probability of success is equal to 86%; these values are obtained by considering the propagation through the system of the initial probability values of failure and success associated with the PhysicalComponentModels of the Tank System.

It is worth noting that, in the proposed solution, there is no direct link with any Requirements Management System. Indeed, the definition and implementation of the requirements are based on the exploitation of the *fulfill* and *requirement* constructs. Each requirement that is manually defined (starting from its definition in a Requirements Management System, such as IBM Doors) is then linked to the system design. Then, the algorithm is able to generate the BN in terms of its structure, status, and probabilities, according to GeNie format.

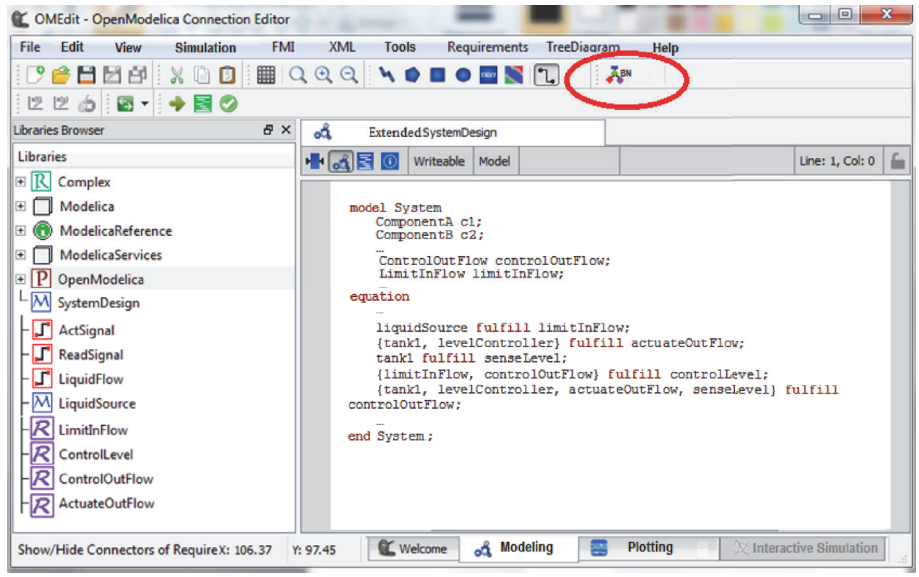

Figure 6: OpenModelica prototype with Modelica language extended for supporting BN generation.

6. Related Work on Dependability Analysis in Modelica

The International Electrotechnical Commission (IEC) defines dependability as "the collective term used to describe the availability performance and its influencing factors: reliability performance, maintainability performance and maintenance support performance." The engineering discipline which aims at providing an integrated and methodological approach to deal with system dependability is commonly indicated by the acronym RAMS (Reliability, Availability, Maintainability, and Safety), whereas the main objective to identify causes and consequences of system failures is called RAMS analyses [38–40].

Facing dependability challenges in Modelica world is a quite young research topic as proved by some recent research works [41]. Indeed, some research efforts are already available in literature for enabling the dependability analysis of system based on the Modelica language and related tools. For example, in [42] a method for detection of the minimal path set of any fault-tolerant technical system that is represented as a multidomain object-oriented model is described, exemplified, and substantiated. Thus, the method, called DMP, automatically performs a safety or reliability analysis of the system. In particular, the proposed minimal path set detection method requires that failure of a system can be simulated in addition to its normal behavior. Thus, the modeling has to be supplemented by equations that reflect failures of system components and, if applicable, by operating logics that determine how a system reacts to the occurrence of component failures. DMP can be employed throughout the system development process to keep the safety analysis up to date with design iterations. This is meaningful particularly if multidomain object-oriented modeling is used already in systems engineering. DMP enhances the scope of application of a model while permitting all other simulation studies that originally motivated implementation of the model to be conducted. DMP belongs to the class of state-space simulations. Evaluation of the system graph reduces the number of simulations required, thus ensuring feasibility of DMP which has been already tested also on large, realistic models of safety relevant aircraft systems, as described in [43].

In [44] a fault injection method for liquid rocket pressurization and feed system (PFS) without modifying the system structure is presented. Firstly, a physics-based model of a pressurization and feed system based on Modelica, which describes both nominal and faulty behaviors in a unified way, is developed. Then, a fault injection method is discussed, which uses the fault mode library and constructs association between the Modelica model and the fault mode by customized Modelica annotation in MWorks. Consequently, several typical fault modes such as leakage and clogging are simulated to verify this method. The result shows that this new method could be easily used to simulate various kinds of fault modes in liquid rocket pressurization and feed system and the new fault simulation process indeed plays a role in the system design and could give some reference to ongoing fault detection and diagnoses.

In [45], the paper presented some results that aimed at developing methods and tools for multidisciplinary collaborative development of dependable embedded systems, by including the modeling of both normal and faulty behavior though the exploitation of the Modelica language and related software tool. The authors focused on the construction and analysis by cosimulation of formal models that combine discrete-event specifications of computer-based controllers with continuous-time models of the environment with which they interact.

Meanwhile [46] focused their attention on air handling unit (AHU) that is the main component of heating, ventilation, and air-conditioning (HVAC) systems, in which irregular faults in AHUs are major sources of energy consumption.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<smile version="1.0" id="Unnamed" numsamples="1000" discsamples="10000">
<nodes>
                    <cpt id="LevelController">
                              <state id="Success"/>
                              <state id="Failure"/>
                              <probabilities>0.9 0.09999999999999998</probabilities>
                    </cpt>
                    <cpt id="Tank">
                              <state id="Success"/>
                              <state id="Failure"/>
                              <probabilities>0.95 0.05000000000000004</probabilities>
                    </cpt>
                    <cpt id="Sink">
                              <state id="Success"/>
                              <state id="Failure"/>
                              <probabilities>0.4 0.6</probabilities>
                    </cpt>
                    <cpt id="ActuateOutFlow">
                              <state id="Success"/>
                              <state id="Failure"/>
                              <parents>Tank</parents>
                              <probabilities>0.5 0.5 0.5 0.5</probabilities>
                    </cpt>
                    <cpt id="SenseLevel">
                              <state id="Success"/>
                              <state id="Failure"/>
                              <parents>Tank</parents>
                              <probabilities>0.5 0.5 0.5 0.5</probabilities>
                    </cpt>
                    <cpt id="LiquidSource">
                              <state id="Success"/>
                              <state id="Failure"/>
                              <probabilities>0.8 0.2</probabilities>
                    </cpt>
                    <cpt id="LimitInFlow">
                              <state id="Success"/>
                              <state id="Failure"/>
                              <parents>LiquidSource</parents>
                              <probabilities>0.5 0.5 0.5 0.5</probabilities>
                    </cpt>
                    <cpt id="ControlOutFlow">
                              <state id="Success"/>
                              <state id="Failure"/>
                              <parents>SenseLevel ActuateOutFlow LevelController</parents>
                              <probabilities>0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5</probabilities>
                    </cpt>
                    <cpt id="ControlLevel">
                              <state id="Success"/>
                              <state id="Failure"/>
                              <parents>ControlOutFlow LimitInFlow</parents>
                              <probabilities>0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5</probabilities>
                    </cpt>
          </nodes>
<extensions> ... </extensions>
</smile>
```

Box 1: A fragment of the generated XML code representing the Bayesian Network.

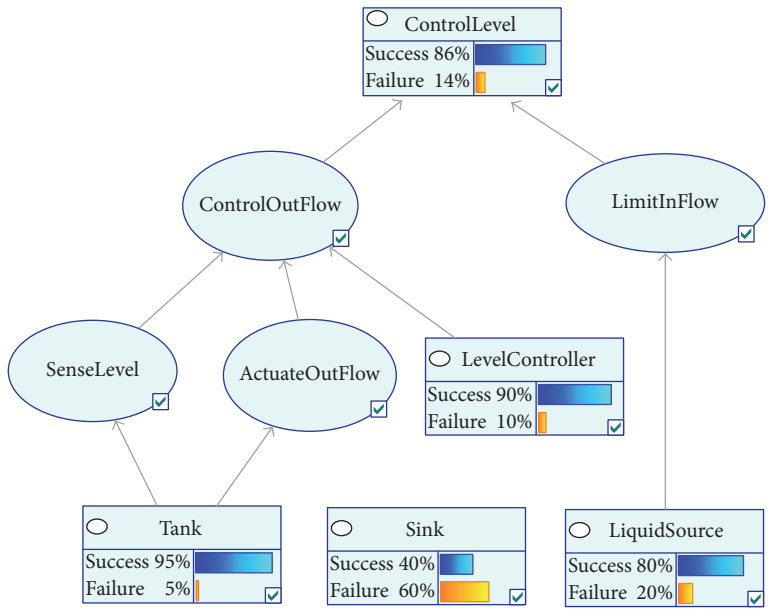

Figure 7: Bayesian Network generated from a Modelica-based model.

Various researches on fault detection and diagnosis in HVAC systems have been published; however, these studies lack inclusion of data on real sensors and disposal of noises. In addition, actual application of data measurement of simple correlation is difficult despite the use of complicated models and methods. For energy efficient operation of HVAC, this research effort aims to detect and diagnose three abnormal states in the AHU with the popular deep learning model, called Deep Belief Network (DBN), where various data generated by Modelica have been used as training set.

Furthermore, many collaborations among industries and academies have great interest in facing dependability issues as proved by several research projects such as EUROSYS-LIB, MODELISAR, OPENPROD, and MODRIO [27, 47–49], which are actively involved in dependability aspects of systems. The major research directions are mainly based on two approaches: one centered on the exploitation of Modelica annotations and the other one based on the implementation of Modelica library. Modelica annotations are attributes that are used to provide additional information associated with some element of a Modelica model, which are typically handled at tool level. Examples of annotations are model documentation and graphical design information concerning icons and connections associated with models. Centered on Modelica annotations is, for example, a toolchain that enables extracting specific information from a Modelica-based design, making them available to be analyzed in a Figaro based environment. Figaro is a probabilistic programming language that supports development of probabilistic models and provides reasoning algorithms that can be applied to models to draw useful conclusions from evidence [50]. Through Figaro it is possible to deduce and extract automatically the data which are necessary for supporting the classic reliability model processing codes such as Fault Tree and Markov Chains [51]. Specifically, in this approach Modelica-based components are annotated with specific information that allows building automatically a Fault Tree and perform fault tree analysis by exploiting Figaro. The main advantage of this approach is that the basic Modelica modeling language does not need to be extended and as a consequence nothing has to be done at the compiler level, whereas in order for such extra information to be used, each single simulation tool needs to be updated in order to support specific function to handle properly such annotations.

A different approach is focused on extensions of the Modelica language. In particular, an important work consists of the definition of formal requirements specification language, called FORM-L, which is mainly intended to be used by application specialists rather than modeling experts [52]. Its objective is to allow the formal modeling of dependable requirements (such as of reliability, safety, and availability) in socio-cyber-physical systems (SCPS) and assumptions regarding its environment. In its most general form, the specification of a requirement in FORM-L addresses four questions: (i) *what* is to be satisfied; (ii) *when* in time that is to be satisfied; (iii) *where* in the system that is to be satisfied; (iv) *how well* the *what* needs to be achieved, as real life systems are bound to have failures.

Based on the FORM-L specification the following main contributions have been recently provided to expressing requirements in Modelica [53], as well as to face properties modeling, as it is explained in [54], where a property is meant to be a condition that must be held true. In particular, in order to improve the engineering processes and the corresponding verification and validation phases, this article deals with the modeling of system properties in a Modelica framework. Specifically, after having introduced several theoretical concepts to formally describe dependable requirements, the development of a dedicated library is explained and illustrated on an industrial example taken from the aeronautics

domain. Similarly, in [55, 56], an open source library for modeling system requirements has been defined, developed, and freely released. Such library, which implements the FORM-L specifications, allows a more formal modeling of requirements as well as their automatic simulation-based verification. The definition and the exploitation of user libraries have the advantages of being easy to use and reusable, even though from the other side it is typically tool-dependent and limited to the available functions. In fact, it is necessary to identify a priori what concepts and aspects the library wants to catch and models. Indeed, a library is conceived for case studies of a specific domain by identifying ad hoc evaluation monitoring metrics such as threshold monitoring, operating domain monitoring, rate of change monitoring, accumulation monitoring, oscillation monitoring, and monitoring with space and time locators. As a consequence, the modeling of new aspects and their related properties (safety, reliability, etc.) mean to extend the library continuously.

A different perspective, as proposed in this current research paper by the authors, is at language level. It brings the advantages of providing new concepts and related keywords that are explicitly devoted to properties modeling and thus high flexibility in the definition of new properties. In particular, a metamodel, which describes the basic concepts for the representation of properties of physical systems, has been appropriately defined and discussed in [57]. Then, specific language extensions for supporting the dependability analysis are introduced and then implemented and prototyped in OpenModelica. This approach enables the definition of system properties regardless of any specific application domain as well as of metrics for their evaluation, even though there is need of extending both the language and the tools that implement the language constructs at compiler level, as well as to manage backward compatibility. It worth noting that a similar approach was proposed successfully for supporting the generation of partial Markov Chain from high level descriptions that allows (i) avoiding the manual construction of Markov Chains, which is both tedious and error prone, as well as (ii) pushing back dramatically the exponential blow-up of the size of the resulting chains [58].

7. Conclusion

The paper has presented a model-driven approach for supporting dependability analysis of a Modelica-based system design through a Probability Model based on the employment of Bayesian Networks. In particular, starting from an already existing Modelica representation, ad hoc constructs are introduced for enriching a physical system design (i) not only to enable the modeling and verification of the system behavior against requirements through simulation in OpenModelica (ii) but also for generating the associated Bayesian Networks for enabling a complementary dependability analysis of the system structure. Specific extensions of the Modelica language (*requirement* and *fulfill*) have been defined and implemented at compiler level, in order to integrate in a Modelica-based system design the proposed Probability Model; then, dedicated OpenModelica APIs have been defined and implemented for generating the Bayesian Network.

The proposed approach combines the benefits of OMedit editors for the system design with GeNie, a dedicated analysis tool. The possibility of combining an already Modelica-based system design model (through such Modelica language extensions) with a Probability Model, allows (i) supporting the analysis of important system properties and (ii) reducing incoherencies and simplifying model modification and reuse. Finally, a prototype of the OpenModelica simulation platform, able to support both the modeling of probabilities and the automatic generation of Bayesian Network for analyzing the system in GeNie, has been experimented on a typical benchmark (a Tank System model), to highlight the effectiveness of the overall proposed model-driven process for dependability analysis.

Furthermore, possible future perspectives are oriented on the introduction of patterns for modeling dysfunctional behaviors and fault injections, which could be used to analyze the behavior of physical components/systems under specific operating conditions and thus evaluating and potentially comparing different design choices through simulation. Moreover, solutions such as semantic-based representation model and FMI for supporting in a wider way reusability and interoperability aspects could be considered. The Modelica extensions described in this paper are under specification in order to be submitted to the Modelica Association as possible extensions for the next release of the Modelica language.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This paper is based on research performed within the ITEA 3 MODRIO (MOdel-DRIven physical systems Operation) Project. Special thanks are devoted to the partners of the Work Package "WP2-Properties Modeling."

References

[1] J. C. Laprie, *Dependability: Basic Concepts and Terminology*, Springer, Berlin, Germany, 1992.

[2] L. Wang, "Get real: real time software design for safety-and mission-critical systems with high dependability," *IEEE Industrial Electronics Magazine*, vol. 2, no. 1, pp. 31–40, 2008.

[3] R. G. Bertodo, "Reliability in automotive design," *International Journal of Vehicle Design*, vol. 9, no. 2, pp. 140–158, 1988.

[4] W. Lim, J. Jang, S. Kim et al., "Reliability-based design optimization of an automotive structure using a variable uncertainty," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 230, no. 10, pp. 1314–1323, 2016.

[5] R. Guillerm, H. Demmou, and N. Sadou, "Engineering dependability requirements for complex systems—a new information model definition," in *Proceedings of the 4th International Systems*

Conference (SysCon '10), pp. 149–152, San Diego, Calif, USA, April 2010.

[6] P. O'Connor, *Reliability Engineering Handbook (Quality and Reliability)*, CRC Press, 1999.

[7] K. Kanoun and M. Ortalo-Borrel, "Fault-tolerant system dependability-explicit modeling of hardware and software component-interactions," *IEEE Transactions on Reliability*, vol. 49, no. 4, pp. 363–376, 2000.

[8] D. Lee and J. Na, "A novel simulation fault injection method for dependability analysis," *IEEE Design & Test of Computers*, vol. 26, no. 6, pp. 50–60, 2009.

[9] R. T. Hessian, B. B. Salter, and E. F. Goodwin, "Fault-tree analysis for system design, development, modification, and verification," *IEEE Transactions on Reliability*, vol. 39, no. 1, pp. 87–91, 1990.

[10] R. McDermott, R. Mikulak, and M. Beauregard, *The Basics of FMEA*, CRC Press, 2009.

[11] J. Che, M. Lv, Z. Yang, Z. Wang, and F. Xu, "Equipment systems reliability analysis based on FTA," in *Proceedings of the International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE '12)*, pp. 293–296, IEEE, Chengdu, China, June 2012.

[12] G. DAmico, F. Petroni, and F. Prattico, "Reliability measures for indexed semi-Markov chains applied to wind energy production," *Reliability Engineering and System Safety*, vol. 144, pp. 170–177, 2015.

[13] S. K. Chauhan and S. C. Malik, "Reliability evaluation of series-parallel and parallel-series systems for arbitrary values of the parameters," *International Journal of Statistics and Reliability Engineering*, vol. 3, no. 1, 2016.

[14] K. S. Son, D. H. Kim, C. H. Kim, and H. G. Kang, "Study on the systematic approach of Markov modeling for dependability analysis of complex fault-tolerant features with voting logics," *Reliability Engineering & System Safety*, vol. 150, pp. 44–57, 2016.

[15] "Space product assurance: methods and techniques to support the assessment of software dependability and safety," Tech. Rep. ECSS-Q80-03, ESA Publications Division, 2006.

[16] R. F. Stapelberg, *Handbook of Reliability, Availability, Maintainability and Safety in Engineering Design*, Springer, Berlin, Germany, 1st edition, 2008.

[17] F. Liang, W. Schamai, O. Rogovchenko, S. Sadeghi, M. Nyberg, and P. Fritzson, "Model-based requirement verification: a case study," in *Proceedings of the 9th International Modelica Conference (Modelica '12)*, Munich, Germany, September 2012.

[18] G. Nicolescu and P. J. Mosterman, *Model Based Design for Embedded Systems (Computational Analysis, Synthesis and Design of Dynamic Systems)*, CRC Press, 2009.

[19] P. David, V. Idasiak, and F. Kratz, "Reliability study of complex physical systems using SysML," *Reliability Engineering and System Safety*, vol. 95, no. 4, pp. 431–450, 2010.

[20] G. Fortino, A. Garro, and W. Russo, "A discrete-event simulation framework for the validation of agent-based and multi-agent systems," in *Proceedings of the WOA 2005—6th AI*IA/TABOO Joint Workshop "From Objects to Agents": Simulation and Formal Analysis of Complex Systems*, pp. 75–84, Camerino, Italy, November 2005.

[21] G. Fortino, A. Garro, W. Russo, R. Caico, M. Cossentino, and F. Termine, "Simulation-driven development of multi-agent systems," in *Proceedings of the 4th International Industrial Simulation Conference (ISC '06)*, pp. 17–24, Palermo, Italy, June 2006.

[22] B. Dodson and D. Nolan, *Practical Reliability Engineering*, John Wiley & Sons, New York, NY, USA, 2002.

[23] Modelica and Modelica Association 2016, https://www.modelica.org/.

[24] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, San Mateo, Calif, USA, 1988.

[25] L. Portinale and A. Bobbio, "Bayesian networks for dependability analysis: an application to digital control reliability," in *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI '13)*, Washington, DC, USA, July 2013.

[26] GeNIe 2016, http://www.openclinical.org/dld_genieSmile.html.

[27] MODRIO 2016 (Model Driven Physical Systems Operation) ITEA 3 Project, https://itea3.org/project/modrio.html.

[28] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3*, Wiley IEEE Press, 2nd edition, 2015.

[29] OpenModelica 2016, https://openmodelica.org/.

[30] Dymola 2016, http://www.3ds.com/products-services/catia/products/dymola.

[31] Wolfram—SystemModeler (MathModelica), http://www.mathcore.com/products/mathmodelica/.

[32] S. Robinson, R. E. Nance, R. J. Paul, M. Pidd, and S. J. E. Taylor, "Simulation model reuse: Definitions, benefits and obstacles," *Simulation Modelling Practice and Theory*, vol. 12, no. 7-8, pp. 479–494, 2004.

[33] E. Gallego, J. M. Alvarez Rodríguez, and J. Llorens, "Reuse of Physical system models by means of semantic knowledge representation: a case study applied to Modelica," in *Proceedings of the The 11th International Modelica Conference*, pp. 747–757, Linköping University Electronic Press, Versailles, France, September 2015.

[34] FMI 2016—Functional Mock-Up Interface v 2.0., https://www.fmi-standard.org/.

[35] P. Fritzson and A. Pop, "Meta-programming and language modeling with MetaModelica 1.0," Tech. Rep. 2011:9, Linköping University, 2011.

[36] OpenModelica Prototype 2016, https://openmodelica.org/svn/OpenModelica/branches/requirements.

[37] A. Tundis, L. Rogovchenko-Buffoni, A. Garro, M. Nyberg, and P. Fritzson, "Performing fault tree analysis of a modelica-based system design through a probability model," in *Proceedings of the International Workshop on Applied Modeling and Simulation (WAMS '13)*, Buenos Aires, Argentina, November 2013.

[38] D. Chambers and M. Chambers, *RAMS Analysis Guides*, Kindle, 2012.

[39] P. J. Kennedy, "Application of RAM to communication systems," *IEEE Transactions on Reliability*, vol. 25, no. 5, pp. 304–310, 1976.

[40] L. Cauffriez, V. Benard, and D. Renaux, "A new formalism for designing and specifying RAMS parameters for complex distributed control systems: the safe-SADT formalism," *IEEE Transactions on Reliability*, vol. 55, no. 3, pp. 397–410, 2006.

[41] M. Bouissou and X. De Bossoreille, "From modelica models to dependability analysis," in *Proceedings of the 5th IFAC International Workshop on Dependable Control of Discrete Systems (DCDS '15)*, pp. 37–43, IFAC-PapersOnLine, Elsevier, May 2015.

[42] C. Schallert, "Automated safety analysis by minimal path set detection for multi-domain object-oriented models," in *Proceedings of the 11th International Modelica Conference*, pp. 565–575, Versailles, France, September 2015.

[43] C. Schallert, *Integrated Safety and Reliability Analysis Methods for Aircraft System Development Using MultiDomain Object-Oriented Models*, 2015.

[44] Z. Mingqing, X. Gang, S. Jinato, C. Lipiing, and Z. Fanli, "A new fault injection method for liquid rocket pressurization and feed system," in *Proceedings of the 11th International Modelica Conference*, pp. 557–563, Versailles, France, September 2015.

[45] J. Fitzgerald, P. G. Larsen, K. Pierce, M. Verhoef, and S. Wolff, "Collaborative modelling and co-simulation in the Development of Dependable Embedded Systems," in *Book Chapter of Integrated Formal Methods*, vol. 6396 of *Lecture Notes in Computer Science*, pp. 12–26, Springer, Berlin, Germany, 2010.

[46] D. Lee, B. Lee, and J. W. Shin, "Fault detection and diagnosis with modelica language using deep belief network," in *Proceedings of the 11th International Modelica Conference*, Versailles, France, September 2015.

[47] EUROSYSLIB 2010—ITEA 2 Project, http://www.eurosyslib.com/.

[48] ODELISAR 2011—ITEA 2 Project, https://itea3.org/project/modelisar.html.

[49] OPENPROD 2012—ITEA 2 Project, https://itea3.org/project/openprod.html.

[50] M. Bouissou, H. Bouhadana, M. Bannelier, and N. Villatte, "Knowledge modelling and reliability processing: presentation of the FIGARO language and associated tools," in *Proceedings of the International Conference on Computer Safety, Reliability and Security (SafeComp)*, Trondheim, Norway, 1991.

[51] M. Bouissou and L. Buffoni, "Dependability analysis for modelica models," in *Proceedings of the International Workshop (MODPROD '15)*, Linkoping, Sweden, 2015.

[52] T. Nguyen, "FORM-L: A MODELICA extension for properties modelling illustrated on a practical example," in *Proceedings of the 10th International Modelica Conference*, pp. 1227–1236, Lund, Sweden, March 2014.

[53] P. F. Lena Buffoni, "Expressing requirements in modelica," *SNE Simulation Notes Europe*, vol. 25, no. 3-4, pp. 185–189, 2015.

[54] A. Jardin, D. Bouskela, T. Nguyen et al., "Modelling of system properties in a modelica framework," in *Proceedings of the The 8th International Modelica Conference*, pp. 579–592, Technical Univeristy, Dresden, Germany, March 2011.

[55] M. Otter, N. Thuy, D. Bouskela et al., "Formal requirements modeling for simulation-based verification," in *Proceedings of the The 11th International Modelica Conference*, pp. 625–635, Linköping University Electronic Press, Versailles, France, September 2015.

[56] A. Garro, A. Tundis, M. Otter et al., "On formal cyber physical system properties modeling: a new temporal logic language and a Modelica-based solution," in *Proceedings of the 2nd IEEE International Symposium on Systems Engineering (IEEE ISSE '16)*, Edinburgh, Scotland, October 2016.

[57] A. Tundis, L. Rogovchenko-Buffoni, P. Fritzson, and A. Garro, "Modeling system requirements in modelica: definition and comparison of candidate approaches," in *Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT '13)*, Nottingham, UK, April 2013.

[58] P.-A. Brameret, A. Rauzy, and J.-M. Roussel, "Automated generation of partial Markov chain from high level descriptions," *Reliability Engineering & System Safety*, vol. 139, pp. 179–187, 2015.