Best of Three Worlds: Towards Sound Architectural Dependability Models

Hichem Boudali Boudewijn R. Haverkort Matthias Kuntz Mariëlle Stoelinga

{*hboudali,brh,kuntzwgm,marielle*}*@cs.utwente.nl*

University of Twente, Department of Computer Science,
P.O. Box 217, 7500AE Enschede, The Netherlands.

Abstract

This paper surveys the most prominent formalisms for availability and reliability analysis and discusses the pros and cons of these approaches. Based on our findings, we outline a solution that unites the merits of the existing approaches into a sound architectural dependability model.

I. Introduction

Dependability evaluation has become an important and integral part in the design of today's computer-based systems. There exists a wide range of techniques and tools for reliability and availability analysis. One may classify these techniques/tools into three broad categories: (1) low-level dependability models, (2) dependability-specific modeling tools, and (3) model-based dependability modeling tools. The first category encompasses general purpose low-level formalisms such as continuous-time Markov chains (CTMC), stochastic Petri nets (SPN) and their extensions, stochastic process algebras (SPA), and input/output interactive Markov chains (I/O-IMC) [1]. The second category consists of formalisms and tools which are specifically geared towards analyzing dependability. In this category, practical tools often define a high-level modeling language, such as (dynamic) fault trees and (dynamic) reliability block diagrams. To carry out the analysis, a low-level model (such as Markov chains or binary decision diagrams) is automatically derived from the dependability-specific model. The third category consists of model-based (i.e., at the system architectural level) formalisms, such as AADL and its error annex and the UML profile for modeling quality of service and fault tolerance characteristics and mechanisms. These approaches annotate an architectural model with dependability properties, from which a dependability-specific model can be (preferably in an automated fashion) generated.

In this paper, we compare these modeling approaches and discuss the shortcomings and problems found in existing formalisms/tools. We divide these issues into (a) modeling effort, (b) expressiveness, (c) semantics and (d) compositionality and modularity. A dependability formalism should enable the dependability analyst to create a model with a reasonable amount of effort, providing appropriate and intuitive (preferably also graphical) syntactic constructs to model the dependability concerns. An important issue here is expressiveness: Does the formalism allow to model all dependability concerns of interest, or does it impose severe restrictions on what can be modeled. Another highly desirable property is that of an unambiguous semantics. Formal semantics pin down the meaning of a dependability formalism in a precise, and unambiguous way and form a rigorous basis for model analysis and tool implementation: Without semantics, dependability models are easily misunderstood, misinterpreted and become unclear and unsound. Finally, we

believe it is important that (dependability) formalisms are compositional. Compositionality means that a model can be built and analysed by dividing it into smaller submodels that are easier to understand and analyze. Compositionality is crucial for making large, complex dependability models understandable and their solutions tractable.

We will see that existing formalisms score very differently on the above criteria. Hence, we propose to unite the merits of the formalisms. Based on our findings, we propose a preliminary solution for a dependability modeling/analysis approach that overarches formalisms in each of the three categories.

The remainder of the paper is organized as follows. In Section II, we provide a brief survey and explain the differences between the three categories of formalisms/tools. In Section III, we explain the main problems and issues found in the formalisms/tools among the various categories. Finally, Section IV sketches a solution for a dependability modeling/analysis approach that tries to combine the merits of each of the three categories.

II. A SHORT SURVEY

This survey is not intended to be comprehensive, its sole purpose is to compare and to provide the reader with a set of representative formalisms/tools that fall into the three categories mentioned above.

1) Low-level dependability models: General purpose formalisms such as CTMCs, SPNs, and SPAs, have been used successfully in dependability studies. These are often considered low-level formalisms, since they do not provide syntactic domain-specific constructs to model dependability. This makes them rather flexible, so that they score well in terms of expressiveness. These models possess a clear semantics and generally (as described below) serve as a semantic domain for other (high-level) formalisms, i.e., the high-level models are translated into these low-level models. Finally, compositionality has been well-studied for some formalisms in this category – compositionality was a major objective for SPAs and I/O-IMCs– whereas others (e.g., CTMCs, SPNs) are not compositional.

2) Dependability-specific modeling tools: Fault trees (FT) [2] and reliability block diagrams (RBD) were among the first high-level graphical modeling techniques developed for reliability/availability system analysis. FTs and RBDs appeal to users because they are simple graphical models whose components (i.e., gates or blocks) directly map to the physical system components being modeled. However, these are combinatorial models where the underlying assumption is that the components are independent. In order to model complex dependencies between components (in the remainder of the paper we will refer to these systems as *dynamic* systems), Markov chains, and their various flavors, have been extensively used in dependability modeling. MCs are indeed a powerful and mathematically sound formalism for explicitly modeling the state space and evolution (i.e., state transitions) of a given system. However, a MC is, as mentioned in above, regarded as a low-level model and building a MC is indeed a tedious and error-prone task. To overcome these problems, modeling languages have been defined where a high-level model is built and then automatically (and transparently to the user) translated into a MC. The system availability estimator [3] (SAVE) modeling language was one of the first such languages. The SAVE language allows to declare components and (failure or repair) dependencies between them using predefined constructs. The SAVE model[1] is then automatically converted into a continuous-time MC (CTMC) for analysis. Many other formalisms/tools have followed this same idea of defining a high-level modeling language in which the user builds his/her model which is then automatically translated into a CTMC or some other intermediate formalism such as SPNs or one of their many extensions. Some of the formalisms/tools that fall into this category are: Dynamic fault trees (DFT) [4], extended fault trees (eFT) [5], dynamic RBD (DRBD) [6], and OpenSESAME [7].

[1]In SAVE, it is also possible to construct purely combinatorial models.

Whereas these are all graphical models, the SAVE tool is only text-based. All these tools have a predefined set of graphical (or textual) constructs which are the building blocks used to create a system dependability model. In fact, these constructs define the basic component behavior and the various failure/repair dependencies between components. We call these constructs the dependability features of the tool. Such features include: spare and primary components (and their management), repair policies, functional dependency (i.e., the failure of one component induces the failure of another component), failure sequence dependency (i.e., a component can only fail after a certain other component fails), fail-over-time (i.e., switching from the primary to the spare component takes a certain amount of time during which the system is not available), etc. Unfortunately, there is no standard set of features and dependability tools often define and use different features. For instance, OpenSESAME uses a fail-over-time feature, whereas DRBDs ignore this feature and assume an instantaneous switching from the primary to the spare component. Nevertheless, reliability engineers agree, to some extent, on a comprehensive set of features that allows them to realistically model any system. One of our goals is to identify such a set of features.

3) Model-based dependability modeling tools: Model (or architecture) based design has gained widespread success in recent years among system developers. Indeed, model-based design has proved to significantly reduce the time and cost of systems' development. Model-based design languages such as the architecture analysis and design language [8] (AADL) or the unified modeling language [9] (UML) are gaining importance in this field. Both AADL and UML have been extended with dependability annotations in [10] and [11] respectively, and some work has been carried out to automatically derive dependability-specific models (such as generalized SPNs) from these extended/annotated AADL [12] or UML [13] models.

III. Problems and issues in existing tools

In this section, we mention some of the concerns and problems with existing formalisms/tools.

a) Expressiveness: On the one hand, formalisms that belong to the first category, such as MCs, are very expressive and powerful from a modeling point of view. On the other hand, dependability-specific modeling tools are restrictive since they often only provide a fixed set of features. For instance, all dependability-specific formalisms/tools define a fixed number of failure modes (or states) for a component, e.g., OpenSESAME defines 3 states: *stand-by, active, and failed*, and compositional models, such as FTs and RBDs define only two states: *operational and failed*, whereas SAVE provides 4 states. In a similar vein, the transitions between these modes is also strictly enforced. It was realized that many applications require a more detailed failure model, e.g., a valve may fail being stuck open or stuck close; a pump may be fully operational, operating at 50% power, operating at 25% power, or failed. To this end extended FTs (eFTs) [5] were defined, allowing arbitrary failure modes. These additional failure modes require, however, more modeling effort in that one has to specify the various conditions under which the mode changes, and the effect of the new failure modes on the overall system failure.

In summary, the modeling power of dependability-specific formalisms varies significantly, and the choice for a particular formalism is often geared by the need to model certain features. Moreover, certain combinations of features are not catered for in any formalism.

Contrary to dependability-specific modeling approaches, architectural (model-based) approaches allow for greater expressiveness. In fact, the user can attach any tailored failure model to a component in the design, where the failure modes and the modes transitions are arbitrarily defined. This increase of modeling power comes at the cost of more modeling effort, and, often, imprecision and ambiguities of the underlying semantics.

b) Semantics: One of the main concerns with higher-level models such as DFTs or AADL (error) models is their semantics. The semantics of these models is often given in terms of low-level models such as CTMCs or SPNs (i.e., the semantic domain). However, the transformation[2]

[2]We will interchangeably use the word transformation, conversion, and mapping.

of the high-level model into its semantic domain often remains unclear and unsound. For instance, in the SAVE tool (we refer the reader to [3] for details), the behavior (thus its semantics) of any basic component is given as a labeled transition system with four states: *Operational*, *Dormant*, *Down*, and *Spare*. It is, however, unclear (1) how a component moves back to its *Spare* state once its primary counterpart component[3] has been repaired, and (2) why there is no transition from the component's *Down* state to the component's *Dormant* state. Another illustration of such imprecise and/or ambiguous semantics is given in the work we have carried out in formalizing the DFT semantics in terms of input/output interactive Markov chains (I/O-IMC) [14], [1]. Indeed, our work and the work by Coppit [15] have revealed some inconsistencies in the DFT semantics which led sometimes to undefined behavior or misinterpretation of the DFT model and therefore produced incorrect results. The problem of unclear and/or incorrect semantics is exacerbated with model-based dependability modeling tools where the formalism is often too expressive and general to pinpoint its correct semantics.

c) Modeling effort: Any formalism/tool should be simple, easy and intuitive to use. In this respect, higher-level and graphical models have a clear advantage over lower-level models, which are only manageable for very small systems. Another reason low-level models are inappropriate is that they are 'flat' models and do not allow hierarchical model-building. Tool support is also an important aspect. In fact, from an engineering point of view, a formalism is useless if it has no adequate tool support. Of course, in order to obtain a correct tool implementation, the formalism needs to have a clear semantics.

d) Compositionality/Modularity: Modularity is a desirable feature for any modeling formalism and can be seen at two different levels: the model-building level and the model-analysis level.

At the model-building level, modularity means that the model is hierarchical and any system can be used as a sub-system in a larger system. A formalism is fully modular at the model-building level if the same behavior at the basic component level is also defined at the complex (composed of multiple interconnected basic components) component level. To illustrate this idea, let's consider DFTs. In DFTs, the most basic component is called a basic event and any basic event can act as a spare. Switching from a spare (backup) mode to a primary mode is done through an activation signal. Unfortunately, this mode switch is only defined for basic events and not for a sub-tree (sub-system). This clearly illustrates a lack in modularity which in turn greatly reduces the modeling power and flexibility of DFTs.

At the model-analysis level, a modeling formalism should allow for decomposition of the model into a set of independent sub-models, solve each of the sub-models separately, and then combine their solutions to produce the overall system solution. This is indeed readily achievable for combinatorial models such as FTs and RBDs. Unfortunately, it is not so simple and often not even possible for dynamic systems. This is due to the nature of the underlying low-level models used in dynamic systems modeling. In fact, the lack of modularity at the analysis level is due to their lack of compositionality. Indeed, for example both MC and SPNs models are either non-compositional or have an informal definition of a composition operation. However, process algebras have a well-defined composition operation, and formalisms that combine process algebras and MCs have been recently defined. Interactive Markov chains and I/O-IMCs are examples of such formalisms and have proved to greatly enhance the modularity at the analysis level [1].

IV. Towards a solution

Based on the discussion above, we outline a solution towards a sound architectural dependability model.

[3]The component it has replaced due to a failure.

In our proposal, we start from a –preferably existing– architectural model: Since such a model is often available during the system design, we avoid duplicating modeling effort. Moreover, the architectural layout will, e.g., through the component dependencies or the number of spares and repair stations, heavily influence the reliability and availability characteristics. Therefore, we believe that the system architecture is the right level to include dependability information.

To overcome the two main drawbacks of existing architectural approaches to dependability (viz. the considerable modeling effort required and the lack of semantics) we envisage the following approach.

First of all, to reduce the modeling effort, we propose to develop standard component dependability models. These could for instance be the four-state SAVE models, or the three-state OpenSESAME models. Just as when using SAVE (or OpenSESAME), the dependability analyst has to specify only the component failure rates, their dependencies and the repair policies; all the other information is present in the standard model. If the designer needs more or different internal component states, tailored component dependability models can be developed, as in the current AADL approach. In this way, the modeling effort is focused on those system parts where the particularities are.

Secondly, we propose to develop rigorous semantics for the developed (standard or tailored) models. We propose to follow the lines of [14]: We present a I/O-IMC model for each component dependability model and each dependability feature in general. We then obtain the semantics of the entire system by composing the component semantics in parallel. At the same time, this approach allows us to analyse the system by compositional aggregation, which turns out to be a powerful technique to combat the state space explosion problem.

Summarizing, we propose to attach a flexible dependability-specific formalism to an (preferably existing) architectural language and to formalize its semantics. In this way, we combine the best of all categories of dependability formalisms/tools.

REFERENCES

[1] H. Boudali, P. Crouzen, and M. Stoelinga, "A compositional semantics for Dynamic Fault Trees in terms of Interactive Markov Chains," accepted to ATVA 2007 conference.

[2] W. E. Veseley, F. F. Goldberg, N. H. Roberts, and D. F. Haasl, "Fault tree handbook, NUREG-0492," NASA, Technical report, 1981.

[3] A. Goyal, W. C. Carter, E. de Souza e Silva, S. S. Lavenberg, and K. S. Trivedi, "The system availability estimator," in *Proceedings of the 16th Int. Symp. on Fault-Tolerant Computing*, July 1986, pp. 84–89.

[4] J. B. Dugan, S. J. Bavuso, and M. A. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems," *IEEE Transactions on Reliability*, vol. 41, no. 3, pp. 363–377, September 1992.

[5] K. Buchacker, "Modeling with extended fault trees," in *Fifth IEEE International Symposium on High Assurance Systems Engineering (HASE 2000)*, Nov 2000, pp. 238–246.

[6] S. Distefano and L. Xing, "A new approach to modeling the system reliability: dynamic reliability block diagrams," in *Reliability and Maintainability Symposium*, Jan 2006, pp. 189–195.

[7] M. Walter, M. Siegle, and A. Bode, "Opensesame: the simple but extensive, structured availability modeling environment," *Reliability Engineering and System Safety*, vol. In Press, corrected proof, April 2007.

[8] "Architecture Analysis and Design Language (AADL)," SAE standards AS5506, Nov 2004.

[9] The Unified Modeling Language, "http://www.uml.org/."

[10] "SAE Architecture Analysis and Design Language (AADL) Annex Volume 1," SAE standards AS5506/1, June 2006.

[11] O. Group, "Uml profile for modeling quality of service and. fault tolerance characteristics and mechanisms," Tech. Rep., june 2006.

[12] A. E. Rugina and K. Kanoun and M. Kaâniche, "An Architecture-based Dependability Modeling Framework Using AADL." Dallas, USA: International Conference on Software Engineering and Applications (SEA2006), Nov 2006.

[13] A. Bondavalli, I. Majzik, and I. Mura, "Automatic dependability analysis for supporting design decisions in UML," in *Proc. of the 4th IEEE International Symposium on High Assurance Systems Engineering*, 1999.

[14] H. Boudali, P. Crouzen, and M. Stoelinga, "Dynamic fault tree analysis using input/output interactive markov chains." in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, UK*. IEEE Computer Society, 2007, pp. 708–717.

[15] D. Coppit, K. J. Sullivan, and J. B. Dugan, "Formal semantics of models for computational engineering: A case study on dynamic fault trees," in *Proceedings of the International Symposium on Software Reliability Engineering*. IEEE, Oct 2000, pp. 270–282.