Arcade - A formal, extensible, model-based dependability evaluation framework [*]

Hichem Boudali[1] Pepijn Crouzen[2] Boudewijn R. Haverkort[1] Matthias Kuntz[1]
Mariëlle Stoelinga[1]

[1] University of Twente, Department of Electrical Engineering, Mathematics and Computer Science
P.O. Box 217, 7500AE Enschede, The Netherlands

[2] Saarland University, Department of Computer Science
Stuhlsatzenhausweg 45, 66123 Saarbrücken, Germany

Abstract

This paper discusses the requirements that a suitable formalism for dependability modeling/evaluation should possess. We also discuss the outline of Arcade*, an architectural dependability formalism that we are developing.*

1 Introduction

Now that computers and communication are proliferating in all kinds of devices and home appliances, requiring high-dependability is not restricted to computers that are being used in traditional "high dependability applications" such as space and aircraft or (nuclear) power control systems. An important difference with these traditional systems, however, is that in modern embedded systems, high dependability is a key concern, but that the costs to be made to achieve it may not be high. Instead, high dependability must be achieved as a "by product" of a sound design and implementation trajectory, almost at no additional costs. This poses several requirements on the modeling and analysis capabilities of a framework for dependability analysis.

In this paper we first discuss the requirements which, in our opinion, a suitable dependability formalism should possess. We also advocate that none of the existing formalisms we know complies with all requirements. Then we lay out our plans for a new, formally well-rooted, and extensible framework for dependability evaluation, that comes very close to a design language: Arcade (for ARChitecturAl Dependability Evaluation). It has been designed so as to combine the strengths of previous approaches and to try to avoid shortcomings. Key features are its formal semantics, compositional modeling *and* analysis, as well as extensibility. In addition, we define our framework in an architectural style, i.e., we define a system model in terms of components or entities that (directly) map to actual physical/logical system components. In fact, our framework is ultimately intended to be incorporated into an architectural design language. Finally, we show through a small example the main features of Arcade.

[*]This research has been partially funded by the Netherlands Organization for Scientific Research (NWO) under FOCUS/BRICKS grant numbers 642.000.505 (MOQS) and 542.000.504 (VeriGem); by the EU under grant numbers IST-004527 (ARTIST2); and by the DFG/NWO bilateral cooperation programme under project number DN 62-600 (VOSS2).

2 Requirements

First, we summarize the requirements which, in our opinion, any good dependability formalism should possess.

1. **Low modeling effort.** A dependability formalism should be simple, easy and intuitive to use, thus enabling the dependability analyst to create a model with a reasonable amount of effort. In this respect, graphical models with clear constructs to model dependability specific concerns have a clear advantage over lower-level models (e.g., state-based), which are only manageable for very small systems.

2. **High expressivity.** A dependability formalism should be able to express all relevant concerns. Clearly, there is a trade off between modeling effort and expressiveness: the more different and/or complex the aspects (or features) a formalism can express, the more complex the formalism becomes. An important requirement of a dependability framework is, therefore, to be extensible thus allowing for future additions of features.

3. **Formal semantics.** Another highly desirable requirement is that of an unambiguous semantics. Formal se-

mantics pin down the meaning of a dependability formalism in a precise and unambiguous way and form a rigorous basis for analysis and tool support: without semantics, dependability models are easily misunderstood, misinterpreted and become unclear and unsound.

4. **Compositionality.** Compositionality (also called modularity) is a key technique to break down the complexity of large systems into smaller and manageable pieces. We distinguish between compositional modeling and compositional analysis. *Compositional modeling* entails that a model can be created by composing smaller submodels. There are two important types of composition: parallel composition, which combines two or more components which are at the same level of abstraction, and hierarchical composition, where one component is internally realized as a combination of subcomponents. *Compositional analysis* means that a model can be analyzed by combining the results of the analysis of the submodels. Compositional analysis is a key feature in combating analysis complexity.

5. **Analysis methods and tool support.** Tool support is another important aspect. In fact, from an engineering point of view, a formalism has little use 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.

3 Existing formalisms

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) general-purpose (dependability) models, (2) dependability-specific modeling tools, and (3) model-based (or architectural) 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 such as stochastic activity networks (SAN), stochastic process algebras (SPA), and input/output interactive Markov chains (I/O-IMC) [4]. In general, these formalisms, specify a system model in terms of states and transitions. This makes them very general (and hence expressive) and precise, but these models are typically large and less structured, hence difficult to understand, since they do not provide any dependability-specific constructs. Some formalisms allow compositional modeling (I/O-IMCs, SPAs, SANs) by a parallel composition operator "$\|$" and/or compositional analysis (I/O-IMCs and SANs), whereas others do not (SPNs).

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 (FTs/DFTs) and (dynamic) reliability block diagrams (RBDs/DRBDs). To carry out the analysis, a low-level model (such as a Markov chain) is automatically derived from the dependability-specific model. Surprisingly, the dependability specific approaches are all somehow limited in expressiveness; although each of them incorporates certain dependability constructs, none of them includes them all. Although we agree that it is impossible to include all possible features, we do think that a modeling approach should be extensible, so as to be able to accommodate any, also future, needs. In earlier work [5], we provided a compositional semantics, analysis methods and tool support for DFTs. Even though a similar approach could be taken for other dependability specific formalisms such as DRBDs, thus relieving our concerns with respect to semantics and compositionality, the lack of expressiveness remains an important issue with these formalisms.

The third category consists of model-based (at the system architectural level) formalisms, such as AADL and its error annex [2], and the UML profile for modeling quality of service and fault tolerance characteristics and mechanisms [7]. Architectural languages require limited modeling effort, since they annotate architectural models (which play an important role throughout the design). However, these languages, as we know them, lack a formal semantics and tool support for automatic dependability evaluation.

Table 1 summarizes this (partially subjective) comparison between the different existing dependability formalisms.

	eff	expres	sem	compos (mod/ana)	tool
general-purpose					
CTMCs	-	+	+	-/-	+
I/O-IMCs	-	+	+	+/+	+
SAN	-	+	+	+/-	+
SPAs	-	+	+	+/+	+
SPNs	-	+	+	-/-	+
specific					
DFTs	+	-	+	+/+	+
DRBDs	+	-	-	+/-	-
model-based					
AADL	+	+	-	+/-	-
UML	+	+	-	+/-	-
Arcade	+	+	+	+/+	+

Table 1. Comparison of dependability evaluation formalisms.

4 The Arcade approach

The aim of our recently started work on Arcade is to unite the strength of existing formalisms, while avoiding their weaknesses.

Key features of Arcade are its architectural approach, reducing the modeling effort; its extensibility, ensuring high expressivity; its formal semantics in terms of I/O-IMCs, which not only pins down the semantics in an unambiguous way, but also enables compositional analysis via the compositional aggregation approach for I/O-IMCs [4]. Below, we describe the Arcade approach and discuss, in depth, how it realizes the requirements on dependability formalisms that we put forward earlier.

4.1 Arcade modeling approach

The basic idea behind Arcade is that it defines a system as a set of interacting components, where each component is provided with a set of operational/failure modes, time-to-failure/repair distributions, and failure/repair dependencies. We propose a predefined set of components along with an extensible set of features (such as interactions, dependencies, operational/failure modes, etc).

We have identified three main components with which we can, in a modular fashion, construct a system model: (1) a Basic Component (BC), (2) a Repair Unit (RU), and (3) a Spare Management Unit (SMU). The underlying semantics of each of these components are I/O-IMCs.

A basic component represents a physical/logical system component that has a distinct operational and failure behavior. A BC can have any number of operational modes (e.g., *active vs. inactive*, *normal vs. degraded*) and can fail either due to an inherent failure (realized as a Markovian transition) or due to a *destructive functional dependency*.

The RU component handles the repair of one or many BCs. Various *repair policies* (e.g., first-come-first-served, priority) and repair dependencies between BCs can be implemented. Finally, the SMU handles the activation and deactivation of BCs used as spare components.

4.2 Requirement fulfillment

Low modeling effort. The Arcade approach requires low modeling effort since its design has been centered around three principles:

1. **Architectural approach.** We advocate that dependability analysis is best done at an architectural level and, more specifically, by annotating existing architectural design models with dependability-specific information. This not only relieves the engineer from the burden of creating new models for the purpose of dependability analysis, but also provides a single model, and thus ensuring integrity, for doing dependability and other design-related evaluation methodologies.

2. **Standard features.** Arcade includes standard features for recurring dependability features. In particular, we provide standard operational/failure modes and behavior, standard repair policies such as dedicated and first-come-first-serve policies, and standard spare management units.

3. **Connect to standard formalisms.** We plan to provide a tight connection of Arcade to existing, graphical formalism such as UML and AADL. In fact, our framework is ultimately intended to be incorporated into an architectural design language.

High expressivity. To balance between expressivity and modeling effort, Arcade is extensible. We provide standard features whenever possible, and allow user-defined features whenever needed. In fact, Arcade provides a standard set of basic components, but also allows the user to define components that, for instance, can exhibit more complex operational/failure modes.

Formal Semantics. We provide a formal semantics of Arcade models in terms of I/O-IMCs: each Arcade component is translated into an I/O-IMC. The semantics of the entire Arcade system model is then obtained by composing in parallel (using the parallel operator "$\|$") the I/O-IMCs of all components.

Compositional modeling and analysis. The Arcade modeling language incorporates both parallel and hierarchical composition. The parallel composition of Arcade components is realized by simply specifying multiple components, saying how one component depends on the operational/failure modes of other components. Hierarchical composition will be realized through interfaces, specifying how the failures of the internal components manifest themselves as failures of the composite component.

On the analysis side, we use the powerful compositional aggregation methods for I/O-IMCs. The I/O-IMC formalism is equipped with several aggressive aggregation (also called lumping or bisimulation minimization) techniques, which replace an I/O-IMC with an equivalent, but smaller I/O-IMC. An important feature is that aggregation is compositional, i.e., one can first aggregate the I/O-IMCs and then compose them together. By performing this procedure in a step-by-step fashion (i.e., take two (lumped) components, compose them, lump the result, compose with another (lumped) component, lump the results, etc), one obtains a state space that is significantly smaller than the state space that is obtained by composing all I/O-IMC models at once.

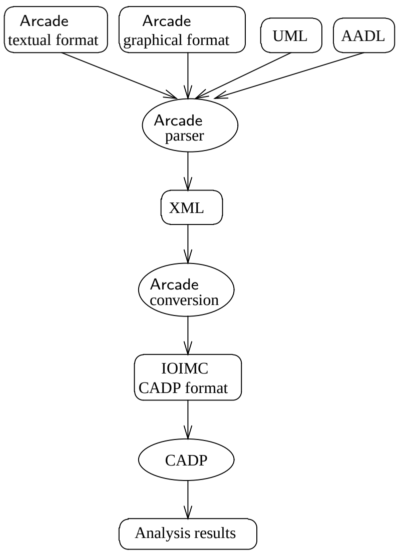

Figure 1. Arcade **tool chain.**

Tool support. We are working on an Arcade analysis tool chain based on the CADP toolset [6], which is a tool for (among others) I/O-IMC analysis and includes methods for I/O-IMC composition, aggregation and analysis. Our tool chain, depicted in Figure 1, takes as input an Arcade model and generates the underlying I/O-IMC models in a format that is readable by CADP. CADP can then compose and minimize the I/O-IMCs based on the compositional aggregation approach, and calculate the desired dependability measures. Currently, only textual input is supported (see example in the next section), but we are planning to develop a graphical language for Arcade models. Moreover, our future plans include a connection of Arcade with UML and AADL.

5 Case study

To demonstrate the feasibility and usability of our approach, we chose a wide-spread case study from the literature, a distributed database system [8]. We briefly describe the system functionality and show parts of its Arcade specification.

5.1 Distributed database system

In [8] a dependability model for a distributed database architecture is described, and as a modeling formalism stochastic activity networks (SANs) [9] were employed.

5.1.1 Verbal description

The system possesses two processors, one of which is a spare. Four disk controllers are divided into two sets. The system has in total 24 hard disks, which are divided in 6 clusters, i.e., each cluster consisting of four disks. Each controller is responsible for three disk clusters, each of the twelve disks the controller set is responsible for, is accessible by any of the two controllers in the respective set. Each processor can access each of the four disk controllers.

The processors are administrated by a spare management unit and share one repair unit. For each disk controller set and disk cluster there is a repair unit responsible. All repair units choose the next item to be repaired according to a first-come first-served (FCFS) repair strategy.

The system is down, if one of the following conditions is met: (1) all processors are down, or (2) in at least one controller set, no controller is operational, or (3) more than one disk in each cluster is down.

5.1.2 Arcade **model**

The Arcade models for the components of the distributed database system are fairly simple. Most components have a unique operational mode, except the spare processor which has two modes (i.e., inactive and active). Below, we describe the system in a textual format. The syntax should be self-explanatory.

1. Arcade model of processor: Here, we have two Arcade models, one for the primary processor, and one for the spare processor:

   (a) Primary processor

   COMPONENT: pp
   TIME-TO-FAILURE: $exp(\frac{1}{2000})$
   TIME-TO-REPAIR: $exp(1)$

   The disk controllers ($dc\_i, i = 1, \cdots, 4$) and the disks ($d_j, j = 1, \cdots, 24$) have the same Arcade model, except for a different time-to-failure in case of the disks, which is $exp(\frac{1}{6000})$.

   (b) Spare processor:

   COMPONENT: ps
   OPERATIONAL MODES: (INACTIVE, ACTIVE)
   TIME-TO-FAILURE: $exp(\frac{1}{2000})$, $exp(\frac{1}{2000})$
   TIME-TO-REPAIR: $exp(1)$

2. Arcade model of processor repair unit: The repair unit for processors is responsible for both the primary and the spare processors. A simple FCFS repair strategy is assumed:

REPAIR UNIT: $p.rep$
COMPONENTS: pp, ps
REPAIR STRATEGY: FCFS

3. Arcade model for the evaluation criteria: The evaluation criteria formalizes the conditions under which the system is down, in terms of a Boolean expression (*Fault tree*). The single failure conditions are expressed in terms of the relevant[1] failure modes of the respective components.

   SYSTEM DOWN:
   $(pp.down \wedge ps.down)$
   $\vee (dc\_1.down \wedge dc\_2.down)$
   $\vee (dc\_3.down \wedge dc\_4.down)$
   $\vee (2of4 \ \ d\_1.down, ..., d\_4.down)$
   $\vee ... \vee (2of4 \ \ d\_21.down, ..., d\_24.down)$

   $(2of4 \ \ d\_1.down, ..., d\_4.down)$ denotes the failure of 2 out of the four disks d_1, d_2, d_3, and d_4.

5.2 Tool support and analysis

5.2.1 Tool support

In principle, for the analysis of Arcade models we have to proceed as follows:

1. At the top-level, we write down the Arcade specifications (as in Sec. 5.1.2).

2. Each Arcade specification of each of the components is translated into its corresponding I/O-IMC.

3. From the components' I/O-IMCs, the I/O-IMC of the overall system is obtained as follows [4]:

   (a) Choose two I/O-IMCs, and compose them in parallel.

   (b) Minimize the thus obtained I/O-IMC.

4. Repeat the previous steps, until a single I/O-IMC (representing the overall system) is obtained.

5. Transform the I/O-IMC into a continuous-time Markov chain (CTMC).

6. Standard techniques [10] can be used on the obtained CTMC to compute dependability measures of interest.

[1]A component can have several failure modes of which not all need to be relevant for the overall system evaluation.

5.2.2 Analysis

Using the methodology described in the previous section we generated the CTMC representing the behavior of the distributed database architecture system. This CTMC has 2,100 states and 15,120 transitions. During the generation of the final model the largest I/O-IMC encountered had 6,522 states and 33,486 transitions. For comparison, the final model generated in [8] had 16,695 states.

Using the system CTMC we can analyze the availability and reliability of the system. Table 2 shows the results of this analysis compared to the SAN-based results in [8]. Note that the reliability results in this table are based on the definition of reliability used in [8], i.e., the probability of having no system failures within a certain mission time assuming that no component is ever repaired. Because of the discrepancy in reliability results we have also analyzed the reliability of the distributed database system with the DFT tool Galileo [1].[2]

Measure	Arcade	SAN	Galileo
A	0.999997	0.999997	-
R(5 weeks)	0.402018	0.425082	0.402018

Table 2. Dependability analysis for distributed database system

6 Summary and conclusions

In this paper we have proposed a new and extensible framework for dependability evaluation: Arcade. Due to its formal underlying model, it can be used compositionally, both for modeling and for analysis purposes. The latter yields great computational advantages, as illustrated in the case studies. Next to that, the Arcade approach is extensible, hence, adaptable to new circumstances or application areas. Furthermore, we see Arcade as an important step towards design languages for large and complex systems. Indeed, the ultimate goal is to integrate Arcade in a design environment, e.g., based on AADL or UML.

It is important to note that although the syntax of the Arcade language bears resemblance to SAVE, the approaches are truly different. Where in SAVE the actual semantics of the models was hidden in software program that coded the translation from that syntax to a large (flat) Markov chain, Arcade has a formal semantical model that allows for compositional evaluation, as well as facilitates the extension of the modeling language.

[2]It is possible to use DFTs here because we do not consider repair.

As for the future, we plan to work on a further automation of the tool chain, as well as connect to design approaches based on AADL and UML. Furthermore, where we now use relatively simple fault-tree like expressions to specify system failure, we plan to allow for CSL-type expressions [3], thus facilitating stochastic model checking of large dependability models.

References

[1] Galileo tool. http://www.cs.virginia.edu/˜ftree.

[2] SAE Architecture Analysis and Design Language (AADL) Annex Volume 1. SAE standards AS5506/1, June 2006. Available at http://www.sae.org/technical/standards/AS5506/1.

[3] C. Baier, B. Haverkort, H. Hermanns, and J. Katoen. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE Trans. Software Eng.*, 29(7):1–18, July 2003.

[4] H. Boudali, P. Crouzen, and M. Stoelinga. A compositional semantics for Dynamic Fault Trees in terms of Interactive Markov Chains. In *Proc. of the 5th International Symposium on Automated Technology for Verification and Analysis*, pages 441–456. LNCS, 2007.

[5] H. Boudali, P. Crouzen, and M. Stoelinga. Dynamic fault tree analysis using input/output interactive markov chains. In *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 708–717. IEEE, 2007.

[6] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2006: a toolbox for the construction and analysis of ditributed processes. In *Proc. of the 19th International Conference on Computer Aided Verification (CAV)*, 2007.

[7] OMG Group. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. Technical report, june 2006.

[8] W. H. Sanders and L. M. Malhis. Dependability evaluation using composed SAN-based reward models. *Journal of Parallel and Distributed Computing*, 15(3):238–254, 1992.

[9] W. H. Sanders and J. F. Meyer. Stochastic Activity Networks: Formal Definitions and Concepts. In *Lectures on Formal Methods and Performance Analysis*, pages 315–343. Springer, LNCS 2090, 2001.

[10] W. J. Stewart. *Introduction to numerical solutions of Markov chains*. 1994.