A Rigorous, Compositional, and Extensible Framework for Dynamic Fault Tree Analysis

Hichem Boudali, *Member*, *IEEE*, Pepijn Crouzen, and Mariëlle Stoelinga

Abstract—Fault trees (FTs) are among the most prominent formalisms for reliability analysis of technical systems. Dynamic FTs extend FTs with support for expressing dynamic dependencies among components. The standard analysis vehicle for DFTs is state-based, and treats the model as a continuous-time Markov chain (CTMC). This is not always possible, as we will explain, since some DFTs allow multiple interpretations. This paper introduces a rigorous semantic interpretation of DFTs. The semantics is defined in such a way that the semantics of a composite DFT arises in a transparent manner from the semantics of its components. This not only eases the understanding of how the FT building blocks interact. It is also a key to alleviate the state explosion problem. By lifting a classical aggregation strategy to our setting, we can exploit the DFT structure to build the smallest possible Markov chain representation of the system. The semantics—as well as the aggregation and analysis engine is implemented in a tool, called CORAL. We show by a number of realistic and complex systems that this methodology achieves drastic reductions in the state space.

Index Terms—Fault trees, reliability, compositionality, formal models, framework.

1 Introduction

Reliability engineering is an important activity in the design of today's computer and communication systems. For safety critical systems, such as airplanes and nuclear power plants, failures can be life threatening; for other applications, such as online ticket vending systems, failures often incur a high cost.

One of the most popular formalisms to model and analyze systems' reliability is the fault tree (FT) formalism [27]. Dynamic fault trees (DFTs) [13], [8], [26] extend standard (or static) FTs by defining additional gates called dynamic gates. These gates allow the modeling of complex system components' behaviors and interactions, which greatly increases the modeling capabilities of the standard FTs. Like standard FTs, dynamic fault trees are a high-level formalism for computing reliability measures of computer-based systems. For over a decade now, DFTs have been experiencing a growing success among reliability engineers.

DFTs, like FTs, describe the system failure in terms of the failure of its components. A DFT is a tree (or rather a directed acyclic graph (DAG), since subtrees can be shared) in which the leaves are *basic events (BEs)* and the other elements are *gates*. A BE typically models the failure of a physical component and is governed by a probability distribution. In this paper, we consider exponential distributions and phase-type distributions, the latter allowing to approximate other probability distributions with arbitrary precision. Gates express how component failures induce system failures and are either static (AND, OR, and the K/M voting gate) or dynamic (Priority AND, SPARE, and the Functional Dependency gate). DFTs are typically used to compute *system unreliability*, that is, the probability that the system fails during a specified period of time (usually called mission time) and under given conditions. Other measures such as the average time until a failure occurs can be computed as well.

Despite their success, current DFT analysis methods have several (mutually related) drawbacks as follows:

1. Existing analysis methods (most notably, the DIFTree method [21] implemented in analysis tools like Galileo [25] and Relex [23]) typically convert a DFT into a continuous-time Markov chain (CTMC) whose states are vectors of modes (up, failed, active, inactive) for each BE. Hence, the size of the state space is exponential in the number of basic events.
2. These methods impose rather severe syntactic restrictions on DFTs, greatly diminishing the modeling flexibility and power of DFTs. Most notably, DFT spare components must be BEs, whereas spare components, in practice, are often entire subsystems.
3. The DFT semantics is rather imprecise and the lack of formality has, in some cases, led to undefined behavior and misinterpretation of the DFT model.
4. DFTs lack comprehensive modular analysis. DIFTree uses a limited form of compositional analysis: it solves in a separate way all stochastically independent subtrees of a static gate, provided none of its ancestors in the tree is a dynamic gate. Then it combines, using Binary Decision Diagrams, their analysis results to obtain the result for the entire DFT. However, this method is not applicable to dynamic gates. In particular, those DFTs whose top node is a dynamic gate cannot be analyzed compositionally.

- *H. Boudali is with the European Space Agency/ESTEC (TEC-QQD), Keplerlaan 1, PO Box 299, 2200 AG Noordwijk ZH, Netherlands. E-mail: hichem.boudali@esa.int.*
- *P. Crouzen is with the Dependable Systems and Software Group, Computer Science Department, Saarland University, Campus Saarbrücken, 66123 Saarbrücken, Germany. E-mail: crouzen@cs.uni-saarland.de.*
- *M. Stoelinga is with the Formal Methods and Tools Group, Department of Computer Science, University of Twente, PO Box 217, 7500 AE Enschede, Netherlands. E-mail: marielle@cs.utwente.nl.*

Manuscript received 15 Feb. 2008; revised 15 June 2009; accepted 7 Sept. 2009; published online 17 Nov. 2009.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSCSI-2008-02-0033.
Digital Object Identifier no. 10.1109/TDSC.2009.45.

5. The current methods are difficult to extend or to modify.

In this paper, we present a framework for DFT analysis based on I/O-IMCs that greatly alleviates these drawbacks. I/O-IMCs are a powerful and versatile formalism to model and analyze stochastic system behavior, and have been used in a number of applications, ranging from telecommunication systems [18] to railway networks [3] and multiprocessor arrays [11]. I/O-IMCs extend CTMCs with input, output, and internal actions, used for communication between several I/O-IMCs. They are equipped with a parallel composition operator, allowing one to build larger I/O-IMCs from smaller ones, and with powerful minimization (a.k.a. lumping) techniques to reduce the state space of an I/O-IMC.

The core of our methodology is a compositional semantics of DFTs in terms of I/O-IMCs. That is, we translate each DFT element (i.e., gate or BE) into one or more I/O-IMCs—obtaining these semantics turned out to be nontrivial, and required a careful reexamination and generalization of the concept of spare activation. Then, the semantics of an entire DFT is obtained as the parallel composition of all DFT element I/O-IMCs. Since these I/O-IMCs semantics pin down the meaning of a DFT in a mathematically precise way, we lift drawback 3 mentioned above. Relatedly, Coppit et al. [10] presented a formal semantics in Z. The main difference between the formal specification in [10] and the formal specification used in this paper is that in our framework, we use a process algebra-like formalism (i.e., I/O-IMC), which comes with two very powerful concepts, namely parallel composition and aggregation/minimization.

Since composing all element I/O-IMCs at once would give the same blowup as the DIFTree method, we use the compositional aggregation method to reduce the size of the models. That is, we compose two I/O-IMCs, hide actions that are no longer needed for communication with other components, and minimize them. We repeat this process until all elements I/O-IMCs have been composed. While this method is still exponential in worst case, our experiments show that serious reductions (one or two orders of magnitude) are realized in practice. Thus, our methodology relieves drawback 1 mentioned above. Since this analysis method is fully compositional, that is, the analysis of a DFT (i.e., its underlying I/O-IMC) is obtained from analysis results of its components (i.e., lumped I/O-IMCs of submodels), we also lift drawback 4.

We note that the order in which we compose these I/O-IMCs matters for the size of the intermediate I/O-IMC models, but not for the final result. We employ heuristics, based on the way individual I/O-IMC models communicate with each other, to obtain smart composition orders. These models have specific properties (acyclicity) that we exploit by tailored algorithms. Also, it turns out that our techniques require much lighter syntactic restrictions than existing DFT methodologies. Any subsystem can now be used as a dependent event and any activation independent subsystem (see Section 4) can be used as a spare component. Hence, drawback 2 is alleviated. Finally, we show how the current DFT semantics can readily be extended or modified; we present extensions with inhibition, mutual exclusion, and repair, thus, addressing drawback 5.

We have implemented our DFT framework in a tool called *CORAL*. The tool derives all I/O-IMC models and composes them using the CADP tool set [15], which is also used to compute the system reliability. We used *CORAL* to analyze nine case studies, including ones showing systems with spare and dependent event subsystems that are currently not supported by any other DFT tool and a system showing the need for nondeterminism. We have compared our tool with Galileo and our experiments show that, in almost all cases, our tool is much faster and generates significantly smaller models (where we consider the largest model encountered during analysis).

This paper combines and extends the work previously carried out in [7] and [5]. In particular, our contributions in this paper are the following:

1. A complete semantics of all DFT elements: Whereas [7] and [5] describe the semantics of DFT gates for a specific number of inputs, we cover here the general case, employing the IOIML notation. Moreover, we allow phase-type distributions as failure distributions of basic events.
2. A complete proof for the congruence theorem.
3. A more extensive set of case studies including systems with spare and dependent event subsystems, which are not currently supported by any other DFT tool and examples showing the need of nondeterminism.
4. *CORAL*, our prototype tool for analyzing DFT using the I/O-IMC semantics. It employs the specialized I/O-IMC minimization algorithm from [12], yielding much faster computation times than [7] and [5].

1.1 Organization of the Paper

The remainder of the paper is organized as follows: In Section 2, we introduce DFTs, and in Section 3, we discuss I/O-IMCs. In Sections 4 and 5, we present the formal DFT syntax and semantics, respectively. In Section 6, we show, through three examples, how one can readily extend the existing DFT formalism. Section 7 presents the compositional aggregation technique and Section 8 describes the *CORAL* tool. Finally, in Section 9, we present a number of case studies, and Section 10 concludes the paper.

2 Dynamic Fault Trees

As described in Section 1, DFTs and FTs are directed acyclic graphs describing the system failure in terms of the failure of its components. Their leaves are labeled with *basic events*, and nonleaves with *gates*.

1. *BE*. A BE, graphically depicted by a circle (see Fig. 1g), typically represents the failure of a basic system component; its failure behavior is governed by a probability distribution. In order to describe these distributions using I/O-IMCs, this paper considers exponential and (acyclic) phase-type distributions, the latter allowing to approximate other probability distributions with arbitrary precision.

   An exponential distribution has a parameter λ that represents the component's failure rate (i.e., number of failures per time unit). A BE has three modes of operation: *dormant*, *active*, and *failed*. In dormant (or standby) mode, the BE failure rate λ is reduced by a

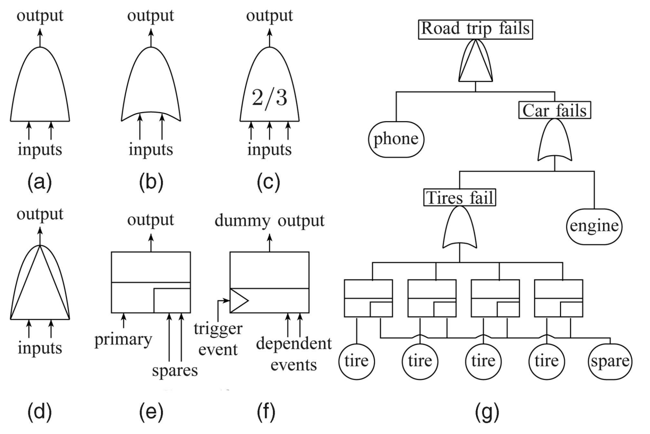


Fig. 1. DFT gates and example. (a) AND gate, (b) OR gate, (c) VOTING gate, (d) PAND gate, (e) SPARE gate, (f) FDEP gate, and (g) DFT example.

factor $\alpha \in [0, 1]$ called *dormancy factor*. Thus, the BE failure rate in standby mode is $\mu = \alpha\lambda$. In active mode, the failure rate is unchanged and equal to λ. The dormancy is relevant when the BE is used as a spare (more details on spare BEs are provided below). In failed mode, the BE, as the name suggests, has failed and remains in that state (i.e., we do not consider repairable systems at this point).

Phase-type basic events (PHBEs) are basic events that fail after a delay governed by a phase-type (PH) distribution [22] with a finite number of phases. The passive behavior of a PHBE is also described by a PH distribution with a finite number of phases. Activation of a PHBE is described by a function, which links passive phases to active phases. When a PHBE is activated, it moves from its current passive phase to the associated active phase. Note that a BE (with exponential distribution) is a special case of a PHBE, where both active and passive distributions have only one phase.

2. *Gates.* Nonleaf elements are called *gates* and express how component failures induce system failures. Their graphical representation is given in Figs. 1a, 1b, 1c, 1d, 1e, and 1f. Each gate has one or more inputs, corresponding to outputs of other elements, and exactly one output. It often represents or maps to a subsystem contained in the whole system, the top element representing the system failure. When the failure event of a BE or a gate occurs, we use the terms failing, occurring, or firing interchangeably.

   Gates can either be static (AND, OR gate, and VOTING (also called K/M) gate) or dynamic. Static gates (which are the only gates in static fault trees) are combinatorial: they are only sensitive to the combinations of failures of their inputs and not to their order.

   Dynamic gates allow the modeling of sequence dependencies (via the priority AND (PAND) gate), functional dependencies (functional dependency (FDEP) gate), and spare management and allocation (via the SPARE gate).[1] Thus, DFTs enrich the FT formalism with powerful and yet easy-to-use modeling capabilities.

   Below, we describe all DFT gates.

3. *AND gate*. The AND gate fails when all of its inputs fail.
4. *OR gate*. The OR gate fails when at least one of its inputs fails.
5. *VOTING gate*. A K/M VOTING gate fails when at least K (called the threshold) out of its M inputs fail.
6. *PAND gate*. The PAND gate fails when all its inputs fail and fail from left to right (as depicted in the figure) order.
7. *FDEP gate*. The functional dependency gate consists of a trigger event (i.e., a failure event) and a set of dependent events. When the trigger event occurs, it causes all the dependent components to become inaccessible or unusable. Essentially, once a dependent component is triggered, it is assumed to have failed. Dependent events, as originally defined in [13], need to be BEs. This restriction will be later lifted in our framework. All dependent events and the trigger event are considered to be inputs to the FDEP gate. The FDEP gate's output is a "dummy" output (i.e., it is not taken into account during the calculation of the system failure probability).
8. *SPARE gate*. The SPARE gate has one primary input and zero (which is a degenerated case) or more alternate inputs called *spares*. The primary input of a SPARE gate is initially powered on (i.e., in active mode) and the alternate inputs are in standby mode. When the primary fails, it is replaced by the first available alternate input (which then switches from standby mode to active mode). This operation is called *spare activation* and causes the spare to switch from dormant to active mode. In turn, when this alternate input fails, it is replaced by the next available alternate input, and so on and so forth. Note that multiple spare gates can *share* a pool of spares. When the primary unit of any of the spare gates fails, it is replaced by the first available (i.e., not failed or not already taken by another spare gate) spare unit, which becomes, in turn, the active unit for that spare gate. The SPARE gate fails when the primary fails and all its spares are failed or unavailable.

If all SPARE inputs are BEs, two special cases arise depending on the spare's dormancy factor α. If $\alpha = 0$, the spare is called a *cold spare* and cannot, by definition, fail before the primary. When $\alpha = 1$, the spare is called a *hot spare* and its failure rate is the same whether in standby or in active mode. If $0 < \alpha < 1$, the spare is called a *warm spare*.

Example. Fig. 1g shows a DFT modeling a road trip. Looking at the top PAND gate, we see that the road trip fails (i.e., we are stuck on the road) if the car fails after the mobile phone has failed; if the car fails first, then we can call the road services to tow the car and continue our journey. The car subsystem fails if either the engine fails or the tires subsystem fails. The car is equipped with a spare tire, which can be used to replace any of the primary tires; when a second tire fails, the tires subsystem fails, causing, in turn, a car failure. Thus, we model the tires subsystem by four spare gates, each having a

1. A fourth gate called "Sequence Enforcing" gate, introduced in [13], can be emulated using a cold spare gate.

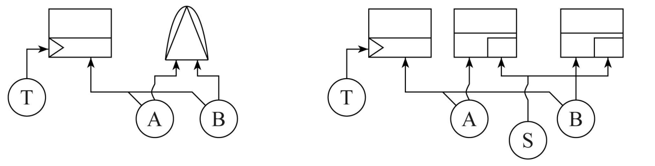


Fig. 2. The occurrence of nondeterminism.

primary tire and all sharing a spare tire. The spare tire is a cold spare, i.e., its failure rate is zero in standby mode.

2.1 Simultaneity and Nondeterminism

In earlier development of the DFT modeling formalism, the semantics (i.e., the model interpretation) of some DFT configurations, where FDEP gates are used, remained unclear. For instance, in Fig. 2, the FDEP gate triggers (in both configurations) the failures of two basic events. Does this mean that the dependent events fail simultaneously, and if so, what is the state of the PAND gate in the left configuration and which spare gate gets the shared spare S in the right configuration? These examples were also discussed in [10], and we believe that this is an inherent nondeterminism in these models. Whereas in [10], these special cases are dealt with by systematically removing the nondeterminism by transforming it into a probabilistic (or deterministic) choice. In our framework, we allow nondeterminism should this be intentional or unintentional. If the nondeterminism was not intended, then its presence (which is easily detected) indicates that an error occurred during the model specification. Nondeterminism could also be an inherent characteristic of the system being analyzed, and therefore, should be explicitly modeled.

In the I/O-IMC formalism, the DFT configurations depicted in Fig. 2 will be interpreted as follows: Whenever the dependent events failure has been triggered, then the trigger event (the cause) happened first and was then immediately (with no time elapsing) followed by the failure of the dependent events (the effect). This adheres to the classical *notion of causality*. Moreover, the dependent events fail in a nondeterministic order (i.e., essentially considering all combinations of ordering). In this case, the final I/O-IMC model is not a continuous-time Markov chain but rather a continuous-time Markov decision process (CTMDP), which can be analyzed by computing bounds of the performance measure of interest [2]. As an example, we have modeled and analyzed a simple nondeterministic case study (see Section 9) using the MRMC model checker [19]. However, the conversion of I/O-IMCs to CTMDPs, which closely follows [17], has not yet been automated in the tool chain.

2.2 Lifting DFT Restrictions

Previously, DFT required all inputs to a SPARE gate and all dependent events of an FDEP gate to be basic events. This restriction greatly diminishes the modeling power of DFTs, since it is very natural to have spare components that are comprised of multiple components or subsystems. To lift this restriction, we need to carefully reexamine the notion of spare activation.

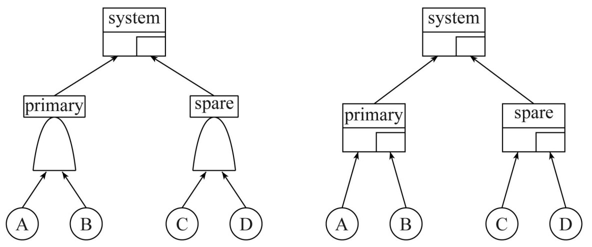


Fig. 3. Complex spare modules.

For primaries and spares that are complex systems, we say that a BE b is a primary BE (or just primary) of a SPARE gate G if b is contained in the subtree that constitutes the primary of G. This is the case if there exists a path from b to G whose last edge ends in the first input of G. Spare-BEs are defined analogously.

The basic idea behind spare activation is that all BEs that are primary-BEs of some SPARE gate are activated from the beginning. A BE that is a spare-BE of some SPARE gate gets activated as soon as one of its SPARE parents is activated. Since spares can be shared, a BE can have multiple SPARE parents.

When a BE is both a primary-BE and a spare-BE, activation is unclear: is this BE activated from the beginning or through the SPARE gate? To rule out such situations, we require all primaries and all spares to be activation-independent subtrees. This means that primaries and spares are disjoint subtrees and that spares can only be shared via their top node.

To illustrate the activation, consider Fig. 3a. Here, the activation of module "spare" simply means the activation of the BEs C and D. The AND gate has the same behavior whether "spare" is active or not. In fact, whenever the SPARE gate (i.e., "system") is activated, it activates BEs A and B.

The behavior of all the non-SPARE gates is unchanged whether they are used as spares or not; the SPARE gate does behave differently when used as a spare. To illustrate this, consider Fig. 3b. When "spare" is not activated (i.e., "primary" has not failed), BEs C and D are dormant; and even if C (being a warm spare) fails, D remains dormant. This is the same behavior as for the "spare" AND gate in Fig. 3a. If now "spare" is activated, the activation signal is only used to activate the primary C and D remains dormant (this is clearly different from the AND gate "spare," where both BEs are activated). Should C fail and "spare" being in its active mode, then D is activated. Thus, "system" activates "spare," while "spare" activates D.

3 Input/Output Interactive Markov Chains

3.1 The I/O-IMC Model

Input/output interactive Markov chains (I/O-IMCs) are a combination of input/output automata (I/O-automata) [20] and interactive Markov chains (IMCs) [16].

I/O-IMCs distinguish two types of transitions: 1) *Interactive transitions* labeled with actions; 2) *Markovian transitions* labeled with rates λ, indicating that the transition can only be taken after a delay that is governed by an exponential

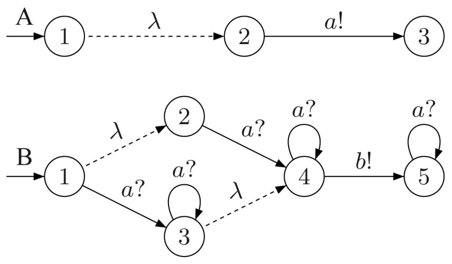


Fig. 4. Two examples of I/O-IMCs.

distribution with parameter λ. Inspired by I/O-automata, actions can be further partitioned into the following:

1. *Input actions* (denoted by $a?$) are controlled by the environment. They can be *delayed*, meaning that a transition labeled with $a?$ can only be taken if another I/O-IMC performs an output action $a!$. A feature of I/O-IMCs is that they are *input-enabled*, i.e., in each state, they are ready to respond to any of their inputs $a?$. Hence, each state has an outgoing transition labeled with $a?$.
2. *Output actions* (denoted by $a!$) are controlled by the I/O-IMC itself. In contrast to input actions, output actions cannot be delayed, i.e., transitions labeled with output actions must be taken immediately. An *observable* action is either an input or an output action.
3. *Internal actions* (denoted by $a;$) are not visible to the environment. Like output actions, internal actions cannot be delayed.

States are depicted by circles, initial states have an incoming arrow without origin, Markovian transitions are denoted by dotted lines, and interactive transitions by solid lines. Fig. 4 shows an I/O-IMC B with two Markovian transitions: one from state 1 to state 2 and one from 3 to 4, both transitions with rate λ. The I/O-IMC has one input action $a?$. To ensure input enabling, we specify $a?$-self-loops in states 3, 4, and 5.[2] Note that state 1 exhibits a race between the input and the Markovian transition: in 1, the I/O-IMC delays for a time that is governed by an exponential distribution with parameter λ, and moves to state 2. If, however, before that delay ends, an input $a?$ arrives, then the I/O-IMC transitions to 3. The only output action $b!$ leads from 4 to 5.

Formally, an I/O-IMC is defined as follows:

Definition 1 (I/O-IMC). *An* input/output interactive Markov chain $\mathcal{P}$ *is a tuple* $\langle S, s^0, A, \rightarrow, \rightarrow^M \rangle$, *where:*

- *S is a set of states.*
- *$s^0 \in S$ is the initial state.*
- *A is a set of discrete actions, where $A = (A^I, A^O, A^{int})$ is partitioned into a set of input actions A^I, output actions A^O, and internal actions A^{int}. This partition is called the* action signature *of $\mathcal{P}$. We write $A^V = A^I \cup A^O$ for the set of visible actions of $\mathcal{P}$.*
- *$\rightarrow \subseteq S \times A \times S$ is a set of interactive transitions. We write $s \xrightarrow{a} s'$ for $(s, a, s') \in \rightarrow$. We require that I/O-IMCs are input-enabled:*

$$\forall s \in S, a? \in A^I \cdot (\exists s' \in S \cdot s \xrightarrow{a?} s').$$

- *$\rightarrow^M \subseteq S \times \mathbb{R}_{>0} \times S$ is a set of Markovian transitions. We write $s \xrightarrow{\lambda}{}^M s'$ for $(s, \lambda, s') \in \rightarrow^M$.*

We denote the components of $\mathcal{P}$ by $S_{\mathcal{P}}$, $s^0_{\mathcal{P}}$, $A_{\mathcal{P}}$, $\rightarrow_{\mathcal{P}}$, $\rightarrow^M_{\mathcal{P}}$, and omit the subscript $\mathcal{P}$ whenever clear from the context.

2. In the sequel, we often omit these self-loops for the sake of clarity and simplicity of the I/O-IMC representation.

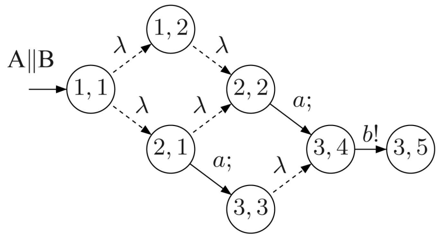


Fig. 5. I/O-IMC ($hide\ a\ in\ A \| B$).

3.2 Parallel Composition and Hiding

The parallel composition operator allows one to build larger I/O-IMCs out of smaller ones. We say that two I/O-IMCs *synchronize* if either 1) they are both ready to accept the same input action or 2) one is ready to output an action $a!$ and the other is ready to receive that same action (i.e., has input action $a?$). I/O-IMCs are also equipped with a parallel composition operator "$\|$," to build larger I/O-IMCs out of smaller ones. The behavior of $\mathcal{P} = \mathcal{Q} \| \mathcal{R}$, i.e., the parallel composition of I/O-IMCs $\mathcal{Q}$ and $\mathcal{R}$ is the joint behavior of its constituent I/O-IMCs and can be described as follows:

1. If an action does not require synchronization (i.e., it belongs to only one of the I/O-IMCs), then $\mathcal{Q}$ and $\mathcal{R}$ can evolve independently, i.e., if $\mathcal{Q}$ (resp. $\mathcal{R}$) can make any transition (interactive or Markovian) and behaves afterward as $\mathcal{Q}'$ (resp. $\mathcal{R}'$), the same behavior is possible in the parallel context, i.e., $\mathcal{Q} \| \mathcal{R}$ can evolve to $\mathcal{Q}' \| \mathcal{R}$ (resp. $\mathcal{Q} \| \mathcal{R}'$).
2. If an action of an interactive transition requires synchronization, then both I/O-IMCs $\mathcal{Q}$ and $\mathcal{R}$ must be able to perform that action at the same time, i.e., $\mathcal{Q} \| \mathcal{R}$ evolves simultaneously into $\mathcal{Q}' \| \mathcal{R}'$. Note that when an output and an input action synchronize, the result is an output action.

Fig. 5 illustrates the parallel composition of I/O-IMCs A and B, where synchronization is on the shared action a. Formally, we have the following:

Definition 2 (Parallel composition). *Let $\mathcal{P}$ and $\mathcal{Q}$ be two I/O-IMCs.*

1. *$\mathcal{P}$ and $\mathcal{Q}$ are* composable *if $A^O_{\mathcal{P}} \cap A^O_{\mathcal{Q}} = A^{int}_{\mathcal{P}} \cap A_{\mathcal{Q}} = A_{\mathcal{P}} \cap A^{int}_{\mathcal{Q}} = \emptyset$.*
2. *If $\mathcal{P}$ and $\mathcal{Q}$ are composable, their* composition *$\mathcal{P} \| \mathcal{Q}$ is the I/O-IMC*

$$(S_{\mathcal{P}} \times S_{\mathcal{Q}}, (s^0_{\mathcal{P}}, s^0_{\mathcal{Q}}), ((A^I_{\mathcal{P}} \cup A^I_{\mathcal{Q}}) \setminus (A^O_{\mathcal{P}} \cup A^O_{\mathcal{Q}}),$$
$$(A^O_{\mathcal{P}} \cup A^O_{\mathcal{Q}}), (A^{int}_{\mathcal{P}} \cup A^{int}_{\mathcal{Q}})), \rightarrow_{\mathcal{P}\|\mathcal{Q}}, \rightarrow^M_{\mathcal{P}\|\mathcal{Q}}),$$

where:

$$\begin{aligned}\rightarrow_{\mathcal{P}\|\mathcal{Q}} = \{&(s,t)\xrightarrow{a}_{\mathcal{P}\|\mathcal{Q}}(s',t) \mid s\xrightarrow{a}_{\mathcal{P}}s' \wedge \\ &a \in A_{\mathcal{P}} \setminus A_{\mathcal{Q}}\} \\ \cup \{&(s,t)\xrightarrow{a}_{\mathcal{P}\|\mathcal{Q}}(s,t') \mid t\xrightarrow{a}_{\mathcal{Q}}t' \wedge \\ &a \in A_{\mathcal{Q}} \setminus A_{\mathcal{P}}\} \\ \cup \{&(s,t)\xrightarrow{a}_{\mathcal{P}\|\mathcal{Q}}(s',t') \mid s\xrightarrow{a}_{\mathcal{P}}s' \wedge t\xrightarrow{a}_{\mathcal{Q}}t' \wedge \\ &a \in A_{\mathcal{P}} \cap A_{\mathcal{Q}}\}\end{aligned}$$

$$\begin{aligned}\rightarrow^{M}_{\mathcal{P}\|\mathcal{Q}} &= \{(s,t)\xrightarrow{\lambda}{}^{M}(s',t) \mid s\xrightarrow{\lambda}{}^{M}_{\mathcal{P}}s'\} \\ &\cup \{(s,t)\xrightarrow{\lambda}{}^{M}(s,t') \mid t\xrightarrow{\lambda}{}^{M}_{\mathcal{Q}}t'\}.\end{aligned}$$

Like in process algebras, the hiding operator hide B in $\mathcal{P}$ makes internal all actions in a set B of output actions such that no further synchronization is possible over actions in B (e.g., in Fig. 5, we hide action a).

Definition 3 (Hiding). *Let $B \subseteq A^O_{\mathcal{P}}$ be a set of output actions. We define hide B in $\mathcal{P}$ as the I/O-IMC given by $(S_{\mathcal{P}}, s^0_{\mathcal{P}}, (A^I_{\mathcal{P}}, A^O_{\mathcal{P}}\backslash B, A^{int}_{\mathcal{P}} \cup B), \rightarrow_{\mathcal{P}}, \rightarrow^M_{\mathcal{P}})$.*

3.3 Weak Bisimilarity

State equivalences, such as bisimulation relations, are crucial in reducing the size of the model to be analyzed. By grouping together equivalent states, one obtains a model that is equivalent but smaller. This operation is called *aggregation*, *lumping*, or *minimization*. For two states s, t to be bisimilar, one requires that all a-transitions in state s can be mimicked in state t. Weak bisimulations abstract from internal computation, thus, the matching transition in t may be a weak transition, consisting of some internal steps, an a step (omitted if a is internal), and some more internal steps. For Markovian transitions, we compare the accumulated rates in s and t.

In this way, bisimilar states have the same observable behavior, and in particular, bisimilar states exhibit the same performance properties.

Our notion of weak bisimilarity for I/O-IMCs generalizes the one for IMCs [16]. Apart from the distinction between input and output transitions, an important difference between our approach and that of [16] is that we ignore Markovian self-loops (as in [9]), which drastically reduces the sizes of the I/O-IMC models.

Let s be a state and $C \subseteq S$ be a subset of states in an I/O-IMC $\mathcal{P}$. We use the following notations:

- The accumulated rate from s into the set of states C is denoted by

$$\gamma_M(s, C) = \sum \{|\lambda \mid s\xrightarrow{\lambda}{}^{M}s' \ \wedge \ s' \in C|\},$$

  where $\{|\cdots|\}$ denotes a multiset of transition rates.
- State s is *stable* if it has no outgoing internal or output transitions.
- $\xrightarrow{\text{int}}$ is the *internal transition relation*, i.e., we have $s \xrightarrow{\text{int}} t$ if $s \xrightarrow{a} t$ for some $a \in A^{int}$. The *weak transition relation* $\Longrightarrow$ arises from $\rightarrow$ by abstracting from internal steps. Thus, we have $s \Longrightarrow t$ if there is a sequence

$$s\xrightarrow{\text{int}}\cdots\xrightarrow{\text{int}}t.$$

  We have $s \overset{a}{\Longrightarrow} s'$ if there exist t, t' such that 1) $s \Longrightarrow t$, $t \xrightarrow{a} t'$ and $t' \Longrightarrow s'$ or 2) $a \in A^{int} \ \wedge \ s \Longrightarrow s'$.
- The set $C^{int} = \{s' \mid \exists s \in C \cdot s' \Longrightarrow s\}$ contains all states with a weak step into set C.

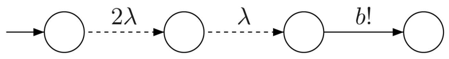


Fig. 6. Aggregation of *hide a in A‖B*.

Definition 4 (Weak bisimulation). *Let $\mathcal{P} = \langle S, s^0, A, \rightarrow, \rightarrow^M\rangle$ be an I/O-IMC. Let R be an equivalence relation on S, then R is a* weak bisimulation *iff for all $(s,t) \in R$, $a \in A$:*

1. *$s \overset{a}{\Longrightarrow} s'$ implies that there is a weak transition $t \overset{a}{\Longrightarrow} t'$ with $(s', t') \in R$.*
2. *$s \Longrightarrow s'$ and s' stable imply that there is t' such that $t \Longrightarrow t'$ and t' is stable and $\gamma_M(s'C^{int}) = \gamma_M(t'C^{int})$, for all equivalence classes $C \in (S/R) \setminus \{[s']_R\}$.*

The states s and t in $\mathcal{P}$ are weakly bisimilar, notation $s \approx_{\mathcal{P}} t$, if and only if there exists a weak bisimulation R with $(s,t) \in R$. Weak bisimilarity for an I/O-IMC$\mathcal{P}$ is defined as the union of all weak bisimulations on $\mathcal{P}$:

$$\approx_{\mathcal{P}} = \bigcup\{R \mid \text{R is a weak bisimulation on } \mathcal{P}\}.$$

We often omit the name of the I/O-IMC if it is clear from context.

The following theorem states that our notion of weak bisimilarity enjoys the expected properties: $\approx_{\mathcal{P}}$ is the largest weak bisimulation relation on $\mathcal{P}$ and weak bisimilarity is a congruence with respect to parallel composition and hiding. Its proof can be found in the Appendix:

Theorem 1. *Let $\mathcal{P}$ and $\mathcal{Q}$ be two I/O-IMCs with identical action signatures, $\mathcal{R}$ be an I/O-IMC composable with $\mathcal{P}$ and $\mathcal{Q}$, and $B \subseteq A^O_{\mathcal{P}}$, then:*

1. *$\approx_{\mathcal{P}}$ is a weak bisimulation on $\mathcal{P}$ and it is the largest weak bisimulation on $\mathcal{P}$.*
2. *$\mathcal{P} \approx \mathcal{Q}$ implies $\mathcal{P}\|\mathcal{R} \approx \mathcal{Q}\|\mathcal{R}$.*
3. *$\mathcal{P} \approx \mathcal{Q}$ implies $\mathcal{R}\|\mathcal{P} \approx \mathcal{R}\|\mathcal{Q}$.*
4. *$\mathcal{P} \approx \mathcal{Q}$ implies hide B in $\mathcal{P} \approx$ hide B in $\mathcal{Q}$.*

Fig. 6 shows the result after applying weak bisimulation on the I/O-IMC resulting from the composition of A and B and the hiding (i.e., made internal) of action a.

3.4 IOIML

IMC modeling language (IML) [16] provides a process algebra-based syntax for specifying IMCs in an easy and concise way. We extend IML to I/O-IMC modeling language (IOIML), which provides a similar syntax for specifying I/O-IMCs. We use IML to describe the semantics of DFT elements in a parametric way.

We assume that there is a countable set of *process variables* V and a countable action signature $A = (A^I, A^O, A^V)$.

Definition 5 (IOIML). *Let $\lambda \in \mathbb{R}_{>0}$, $a \in A$, and $X \in V$. We define the language IOIML as the set of expressions given by the following grammar:*

$$E ::= \mathbf{0} \mid a.E \mid (\lambda).E \mid E + E \mid X \mid \underline{_{x:=}E} \mid \perp .$$

The intuitive meaning of the language constructs is described below:

- The terminal symbol $\mathbf{0}$ describes a *terminated* behavior i.e., the process $\mathbf{0}$ cannot perform any output or internal actions and absorbs all inputs of the I/O-IMC.
- The expression $a.E$ may interact on action a and afterward behave as expression E. We say that E is *action prefixed* by a. As before, we postfix actions with "?", "!", or ";" according to their role as inputs, outputs, or internal actions.

The remaining constructs are identical to their IML counterparts as follows:

- The expression $(\lambda).E$, a *delay prefix* expression, describes a behavior that will behave as expression E after a delay that is governed by an exponential distribution with a mean duration of $1/\lambda$ time units.
- The expression $E + F$ describes two alternatives. It may either exhibit the behavior of expression E or the behavior of expression F.
- The expression $\underline{\scriptstyle X:=}E$ describes a recursively defined behavior. Assuming that the variable X appears somewhere inside expression E, the meaning is as follows: Whenever the variable X is encountered during the evolution of the expression, the expression will reinitialize its behavior to $\underline{\scriptstyle X:=}E$.
- The symbol $\perp$ is intended to represent an *ill-defined* behavior. We will not use this symbol, but it is included for completeness.

The formal semantics of an IOIML expression, i.e., its underlying I/O-IMC, can be obtained in a way similar to the semantics for IML (see [16]). Since the I/O-IMC $\mathcal{P}$ obtained in this way need not to be input-enabled, we complete the expression by adding self-loops $s \xrightarrow{a?} s$ whenever $a?$ is not enabled from state s. An IOIML expression, therefore, must be accompanied by the action signature of the I/O-IMC it describes to be meaningful.

The IOIML description of the I/O-IMC B in Fig. 4 is

$$P1 = (\lambda).P2 + a?.P3, \quad P3 = (\lambda).P4,$$
$$P2 = a?.P4, \qquad\qquad P4 = b!.\mathbf{0}.$$

3.5 IMCs versus I/O-IMCs

IMCs only distinguish between observable and internal actions. All observable actions are delayable and communication is a handshake, i.e., synchronization on action a only occurs when both IMCs involved are ready to perform the a action. While IMCs could in principle be used to model DFTs, we obtain more natural and more concise models by introducing an I/O distinction: it is always the failing DFT element that takes the initiative to notify its failure to its parents in the DFT.

4 DFT Syntax

To formalize the syntax of a DFT, we first define the set $\mathcal{E}$, characterizing each DFT element by its type, number of inputs, and possibly some other parameters. We use the following notations: Given a set X, we denote by $\mathcal{P}(X)$ the power set over X and by X^* the set of all sequences over X. For a sequence $x \in X^*$, we denote by $|x|$ the length of the sequence (also called list) and by $(x)_i$ the ith element in x.

Definition 6. *The set $\mathcal{E}$ of DFT elements consists of the following tuples. Here, $k, n \in \mathbb{Z}^{\geq 0}$ are natural numbers with $1 \leq k \leq n$ and $\lambda, \mu \in \mathbb{R}^{>0}$ are rates:*

- *(OR, n), (AND, n), $(PAND, n)$ represent, respectively, OR, AND, and PAND gates with n inputs.*
- *(VOT, n, k) represents a voting gate with n inputs and threshold k.*
- *$(SPARE, n)$ represents a SPARE gate with one primary and $n - 1$ spares. By convention, the first nondummy input to the SPARE gate is the primary component.*
- *$(FDEP, n)$ represents an FDEP gate with 1 trigger input event and $n - 1$ dependent input events. By convention, the first nondummy input to the FDEP gate is the trigger event.*
- *$(BE, 0, \lambda, \mu)$ represents a BE, which has no inputs (i.e., $n = 0$), an active failure rate λ, and a dormant failure rate μ.*
- *$(PHBE, 0, \phi_A, Q_A, \phi_P, Q_P, \psi)$ represents a phase-type BE, which has no inputs (i.e., $n = 0$), an active failure distribution with $\phi_A \in \mathbb{Z}^{\geq 0}$ phases and generator matrix $Q_A \in \mathbb{R}^{\phi_A \cdot \phi_A}$, and a dormant failure distribution with $\phi_P \in \mathbb{Z}^{\geq 0}$ phases and generator matrix $Q_P \in \mathbb{R}^{\phi_P \cdot \phi_P}$. The activation of the PHBE is described by the function $\psi : [1, \ldots, \phi_P] \to [1, \ldots, \phi_A]$.*

Given a tuple $e \in \mathcal{E}$, we write $type(e)$ for the first item in e, and $arity(e)$ for the second.

We introduce several notions for graphs (potentially with cycles) whose nodes are labeled with DFT elements. An edge in such graphs from v to w means that the output of the DFT element associated with v is an input to the DFT element of w. Since the order of inputs to a gate matters (e.g., for a PAND gate), the inputs to v are given as a list $in(v)$, rather than as a set.

Definition 7. *An element-labeled graph is a triple $\mathcal{D} = (V, in, l)$, where*

- *V is a set of vertices.*
- *$in : V \to V^*$ is an input function that assigns to each vertex a list of inputs.*
- *$l : V \to \mathcal{E}$ is a labeling function that assigns to each vertex a DFT element.*

We write $type(v)$ for $type(l(v))$ and $arity(v)$ for $arity(l(v))$.

Given in, we define the set of edges E_{in} by $\{(v, w) \in V^2 | \exists i. v = (in(w))_i\}$. Thus, E_{in} contains all pairs (v, w) such that v appears as an input of w. We also define the *pruned* input function in' that contains only the nondummy connections between vertices (recall that the outputs of FDEP gates are dummy outputs). Thus, $in'(v) : V \to V^*$ is the function $in'(v)$ that arises from $in(v) = v_1 v_2 \ldots v_n$ by removing all elements v_i s.t. $type(v_i) = FDEP$. Consequently, the set of edges $E_{in'}$ is the set $\{(v, w) \in V^2 | type(v) \neq FDEP \wedge \exists i. v = (in(w))_i\}$ containing all pairs (v, w) such that v appears as a nondummy input of w. Finally, for the sake of spare activation, we define another pruned input function in'' that ignores all inputs, except the trigger input, of all $FDEP$ gates. Thus, $in''(v) : V \to V^*$ is the function $in''(v)$ that arises from $in(v) = v_1 v_2 \dots v_n$ by keeping only the first element (i.e., the trigger) v_1 for all $v \in V$ s.t. $type(v) = FDEP$. Consequently, the set of edges $E_{in''}$ is the set

$$\{(v,w) \in V^2 | \exists i.(type(w) \neq FDEP \land v = (in(w))_i) \lor (type(w) = FDEP \land v = (in(w))_1)\}.$$

We write E for E_{in}, E' for $E_{in'}$, and E'' for $E_{in''}$ if in is clear from the context.

Given a DFT node v, the subtree below v consists of all vertices with a path in E'' leading to v. Node v is *activation independent* if there are no edges leading from $stb(v)$ to a node outside $stb(v)$, except for the outgoing edges of v.

Definition 8. *Let $\mathcal{D}$ be a DFT and $v \in V$ a node in $\mathcal{D}$.*

- *Then* subtree below v, *denoted by $stb(v)$, is the set*
$$\{w \mid \exists v_0, v_1 \dots v_n \in V, n \geq 0. v_0 = w, v_n = v \land \forall 0 \leq i < n.(v_i, v_{i+1}) \in E''\}.$$
- *Vertex v is* activation independent *if $\forall w \in stb(v)$, $w' \in V \setminus stb(v).(w,w') \in E'' \implies w = v$.*

Note that in the definition of an activation-independent vertex, we ignore the inputs, except the trigger input, to $FDEP$ gates as, by convention, activation signals do not propagate through these edges. In the sequel, we will generally refer to an activation-independent vertex as simply an independent vertex.

Finally, we define a DFT as an element-labeled graph $\mathcal{D}$ with several restrictions. These restrictions, which are checked syntactically by our tool, ensure that the DFT contains no anomalies and that it has a well-defined semantics.

Definition 9. *A DFT is an element-labeled graph $\mathcal{D}$ with the following restrictions:*

- *(V, E') forms a directed acyclic graph.*
- *All inputs to a DFT element must be connected to some node in $\mathcal{D}$, i.e., for all $v \in V$, we have $arity(v) = |in(v)|$.*
- *All DFT gates must have at least one nondummy input:*[3] *for all $v \in V$ with $type(l(v)) \neq BE$ and $type(l(v)) \neq PHBE$, we have $|in'(v)| \geq 1$.*
- *There is a unique top element in $\mathcal{D}$, i.e., a non-FDEP element whose output is not connected. That is, there exists a unique $v \in V$, $type(v) \neq FDEP$, such that there is no $w \in V$ with $(v,w) \in E$. This unique v is denoted by $T_{\mathcal{D}}$ or by T if $\mathcal{D}$ is clear from the context.*
- *The first nondummy input of a SPARE gate (i.e., its primary) cannot be an input to another SPARE gate, i.e., primary components cannot be shared: If $v = (in'(w))_1 = (in'(w'))_i$ and $type(w) = type(w') = SPARE$, then $w = w'$.*
- *Nondummy inputs (primary and spare components) to a SPARE gate must be outputs coming from activation-independent vertices (see Section 5 for details): for all $(v,w) \in E'$ with $type(w) = SPARE$, we have that v is activation independent.*
- *An output cannot be twice or more the input of the same gate: For all $w \in V$ and $1 \leq i,j \leq |in(w)|$ with $(in(w))_i = (in(w))_j$, we have $i = j$.*

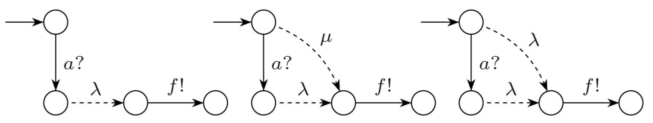


Fig. 7. The I/O-IMCs $[\![(BE,0,\lambda,0)]\!]_{\mathrm{ELT}}(a,f)$, $[\![(BE,0,\lambda,\mu)]\!]_{\mathrm{ELT}}(a,f)$, and $[\![(BE,0,\lambda,\lambda)]\!]_{\mathrm{ELT}}(a,f)$, modeling the semantics of a cold, warm, and hot BE.

5 DFT Semantics

In this section, we first define the semantics of the DFT elements by giving the I/O-IMC for each of the tuples in $\mathcal{E}$. We also need two auxiliary I/O-IMCs: the activation auxiliary, which activates BEs and SPARE gates when they change from dormant to active mode, and the firing auxiliary that handles the dependencies between events as modeled by the FDEP gate. Then, we obtain the semantics of the whole DFT from the parallel composition of the semantics of its elements and the auxiliaries.

The semantics of each non-FDEP element in $\mathcal{E}$ (denoted by $[\![\dots]\!]_{\mathrm{ELT}}$) is a function, which takes as input a number of actions and returns an I/O-IMC. The FDEP gate is handled through the use of firing auxiliaries. We present the graphical descriptions for BEs and gates with two or three inputs and we use the language IOIML to specify the semantics for the general case.

Basic event I/O-IMC model. As pointed out in Section 2, a BE has a different failing behavior depending on its dormancy factor. Fig. 7 shows the (parametrized) I/O-IMCs associated with a cold, warm, and hot BE,[4] i.e., it shows the functions $[\![(BE,0,\lambda,\mu)]\!]_{\mathrm{ELT}} : A^2 \to \mathrm{IOIMC}$ taking as arguments an activation signal $a?$ and a firing signal $f!$.

In IOIML, the I/O-IMC $[\![(BE,0,\lambda,\mu)]\!]_{\mathrm{ELT}}(a,f)$ has action signature $(\{a\},\{f\},\emptyset)$ and is described by the following expression E_0:

$$E_0 = \begin{cases} a?.E_1 + (\mu).E_2, & \text{if } \mu > 0, \\ a?.E_1, & \text{otherwise,} \end{cases}$$
$$E_1 = (\lambda).E_2,$$
$$E_2 = f!.\mathbf{0}.$$

Phase-type basic event I/O-IMC model. A PHBE does not fail after an exponential delay, but rather after a delay governed by a phase-type (PH) distribution [22]. Here, the phase-type distributions for failure in either passive or active mode are described by absorbing CTMCs. A PHBE is described by the tuple $(PHBE, 0, \phi_A, Q_A, \phi_P, Q_P, \psi)$, where $\phi_A, \phi_P \in \mathbb{Z}^{\geq 0}$ denote the number of phases of the active and passive PH distributions, respectively. Matrices $Q_A : [1,\phi_A] \times [1,\phi_A] \to \mathbb{R}$ and $Q_P : [1,\phi_P] \times [1,\phi_P] \to \mathbb{R}$ are the generator matrices of the PH distributions.[5] Finally, the function $\psi : [1,\phi_P] \to [1,\phi_A]$ matches passive phases to

3. Otherwise, the gate should be removed.

4. The hot BE I/O-IMC can be reduced to: $\odot \xrightarrow{\lambda} \bigcirc \xrightarrow{f!} \bigcirc$.

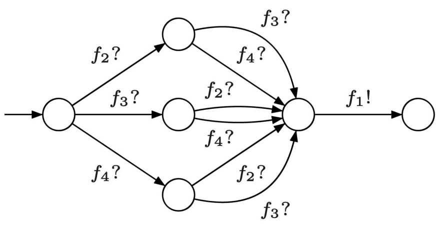


Fig. 8. The I/O-IMC $[\![(VOT, 3, 2)]\!]_{\mathrm{ELT}}(f_1, f_2, f_3, f_4)$.

active phases. If the basic event is activated while its passive failure distribution is in phase i, then the I/O-IMC will move to phase $\psi(i)$ of the active failure distribution. In the case of a cold spare, the number of passive phases is set to 1 with the only entry in Q_p being 0. This is interpreted as being the PH representation with a single state that cannot reach the absorbing state. This representation, in fact, does not represent a true PH distribution, but the semantics is clear: the spare can never fail when it is in passive mode. For other PHBEs, the generator matrices must have strictly negative numbers on the diagonal and positive numbers elsewhere. Furthermore, the sum of each row must be negative.

In IOIML, we find action signature $(\{a\}, \{f\}, \emptyset)$ for the I/O-IMC $[\![(PHBE, 0, \phi_A, Q_A, \phi_P, Q_P, \psi)]\!]_{\mathrm{ELT}}(a, f)$. The I/O-IMC is described by the expression $E_{P,1}$, below $i \in [1, \phi_P]$ and $k \in [1, \phi_A]$:

$$E_{P,i} = \begin{cases} a?.E_{A,\psi(i)}, & \text{if } Q_P(i,i) = 0, \\ a?.E_{A,\psi(i)} + \\ \sum_{1 \le j \le \phi_P \wedge i \ne j} (Q_P(i,j)).E_{P,j} + \\ \Big(-\sum_{1 \le j \le \phi_P} Q_P(i,j) \Big).E_F, & \text{otherwise,} \end{cases}$$

$$E_{A,k} = \sum_{1 \le j \le \phi_A \wedge k \ne j} (Q_A(k,j)).E_{A,j} + \Big(-\sum_{1 \le j \le \phi_A} Q_A(k,j) \Big).E_F,$$

$$E_F = f!.\mathbf{0}.$$

VOTING gate I/O-IMC model. Fig. 8 shows the semantics of the voting gate $(VOT, 3, 2)$ element, i.e., the function $[\![(VOT, 3, 2)]\!]_{\mathrm{ELT}} : A^4 \to \mathrm{IOIMC}$, taking as arguments the output and three input signals of the VOTING gate. The voting gate fires (action f_1) when at least two of its inputs fire (actions f_2, f_3, and f_4).

To define the semantics of a (VOT, n, k) gate with n inputs and threshold k, we use the process variables $P_V(I, U, f, k)$, that depend on three parameters; a set I containing the firing signals of all inputs to the VOT gate, a set U containing the firing signals of inputs that are still operational, and an action f, $f \notin I \cup U$, being the VOT gate's own output firing signal. We set

$$P_V(I, U, f, k) = f!.\mathbf{0} \qquad \text{if } |I \setminus U| \ge k,$$
$$P_V(I, U, f, k) = \sum_{a \in I} a?.P_V(I, U \setminus \{a?\}, f, k) \qquad \text{if } |I \setminus U| < k.$$

5. As the initial distribution of our phase-type representation, we always use the vector $[1, 0, \ldots, 0]$, i.e., a single starting state. This is not a problem since any PH representation can easily be transformed into a PH representation with the same number of phases and a single starting state.

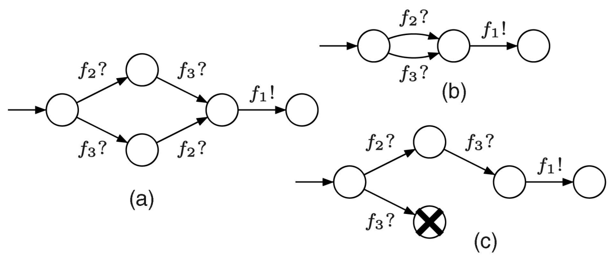


Fig. 9. (a) $[\![(AND, 2)]\!]_{\mathrm{ELT}}(f_1, f_2, f_3)$, (b) $[\![(OR, 2)]\!]_{\mathrm{ELT}}(f_1, f_2, f_3)$, and (c) $[\![(PAND, 2)]\!]_{\mathrm{ELT}}(f_1, f_2, f_3)$.

Thus, $P_V(I, U, f, k)$ emits the failure signal $f!$ after having received k failure signals. The I/O-IMC of an n-input voting gate is: $[\![(VOT,\ n,\ k)]\!]_{\mathrm{ELT}}(f_o, f_1, \ldots, f_n) = P_V(\{f_1, \ldots, f_n\}, \{f_1, \ldots, f_n\}, f_o, k)$ with action signature $(\{f_1, \ldots, f_n\}, \{f_o\}, \emptyset)$. Note that the VOT gate[6] does not have an activation signal as this element does not exhibit a dormant or active behavior as such.

AND gate I/O-IMC model. Fig. 9a shows the semantics of the $(AND, 2)$ gate, i.e., the function $[\![(AND, 2)]\!]_{\mathrm{ELT}}$: $A^3 \to$ IOIMC, taking as arguments the output and two input signals of the AND gate. This I/O-IMC models the fact that the AND gate fires (action f_1) after it receives firing signals from both its inputs (actions f_2 and f_3).

The semantics of an (AND, n) gate with n inputs is defined as a special case of the VOT gate, where the threshold is equal to the number of inputs. The I/O-IMC associated with an n-ary AND gate is then given by:

$$[\![(AND, n)]\!]_{\mathrm{ELT}}(f_o, f_1, \ldots, f_n)$$
$$= P_V(\{f_1, \ldots, f_n\}, \{f_1, \ldots, f_n\}, f_o, |\{f_1, \ldots, f_n\}|)$$

with action signature $(\{f_1, \ldots, f_n\}, \{f_o\}, \emptyset)$.

OR gate I/O-IMC model. Fig. 9b shows the semantics of the OR gate $(OR, 2)$ element, i.e., the function $[\![(OR, 2)]\!]_{\mathrm{ELT}}$: $A^3 \to$ IOIMC, taking as arguments the output and two input signals of the OR gate. The OR gate fires (action f_1) after it receives one of its input firing signals (actions f_2 or f_3).

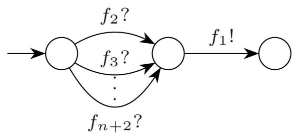
The semantics of an (OR, n) gate with n inputs is defined as a special case of the VOT gate with threshold equal to 1. The I/O-IMC associated with an n-ary OR gate is then given by:

$$[\![(OR, n)]\!]_{\mathrm{ELT}}(f_o, f_1, \ldots, f_n)$$
$$= P_V(\{f_1, \ldots, f_n\}, \{f_1, \ldots, f_n\}, f_o, 1)$$

with action signature $(\{f_1, \ldots, f_n\}, \{f_o\}, \emptyset)$.

PAND gate I/O-IMC model. Fig. 9c shows the semantics of the PAND gate $(PAND, 2)$ element, i.e., the function $[\![(PAND, 2]\!]_{\mathrm{ELT}} : A^3 \to$ IOIMC, taking as arguments the output and two input signals of the PAND gate. The PAND gate fires (action f_1) after all its inputs (actions f_2 or f_3) fire from left to right order. If the inputs fire in the wrong order, the PAND gate moves to an operational absorbing state (denoted by **X**). The semantics of a $(PAND, n)$ gate with n inputs is defined by means of the process variables $P_P(U, f)$. However, now, U is given as a sequence of firing signals of operational inputs, rather than a set, and f is the

6. This is true for all gates except the SPARE gate.



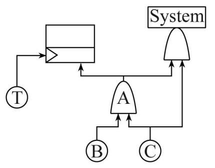


Fig. 10. $FA(f_1, f_2, \{f_3, f_4 \ldots, f_{n+2}\})$ (left) and an example of the FDEP gate extension (right).

$PAND$ gate's own firing signal. The actions in U must occur in the correct order for the PAND gate to fail. We write $U = a_1a_2 \ldots a_n$. We set

$$\begin{aligned} P_P(U, f) &= f!.\mathbf{0} && \text{if } U = \epsilon, \\ P_P(U, f) &= a_k?.P_P(U \setminus \{a_k?\}, f) + \\ &\sum_{a \in U \setminus \{a_k?\}} a?.\mathbf{0} && \text{if } U = a_k a_{k+1} \ldots a_n. \end{aligned}$$

Now $P_P(U, f)$ emits the failure signal $f!$ after having received failure signals from all its inputs, which can only happen if they occurred in the specified order, since deviations from this order end in $\mathbf{0}$. We set

$$[\![(PAND, n)]\!]_{\mathrm{ELT}}(f_o, f_1, \ldots, f_n) = P_P(f_1 \ldots f_n, f_o)$$

with action signature $(\{f_1, \ldots, f_n\}, \{f_o\}, \emptyset)$.

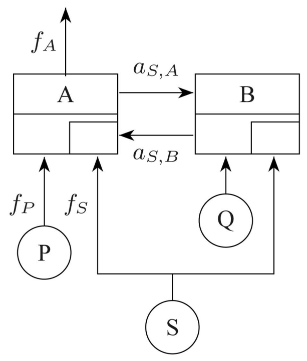
FDEP gate I/O-IMC model. An FDEP gate does not have a semantics in itself, but instead, it is used in combination with the semantics of its dependent events. To model a functional dependency, we define the *firing auxiliary* function $FA : A^2 \times \mathcal{P}(A) \rightarrow$ IOIMC. This (parametric) I/O-IMC ensures that a dependent event fires either when the event fails by itself or when its failure is triggered by the FDEP gate trigger: Fig. 10a shows the FA to be applied in combination with an event that is functionally dependent on n triggers. Signal f_2 corresponds to the failure of the dependent event by itself; signals $f_3, f_4, \ldots, f_{n+2}$ correspond to the failures of any of the triggers; and f_1 corresponds to the failure of the dependent event when also considering its functional dependency upon the triggers. Hence, f_1 is emitted as soon as any signal from $\{f_2, f_3, \ldots, f_{n+2}\}$ occurs. Thus, FA takes as arguments two firing signals and a set of firing signals (corresponding to all triggers of the dependent event).

The I/O-IMC $FA(f_1, f_2, T)$ can, in fact, be interpreted as an OR gate:

$$FA(f_1, f_2, T) = [\![(OR, |T| + 1)]\!]_{\mathrm{ELT}}(f_1, f_2, t_1, \ldots, t_n),$$

where $T = \{t_1, \ldots, t_n\}$. The I/O-IMC $FA(f_1, f_2, T)$ has the following action signature: $(\{f_2, t_1, \ldots, t_n\}, \{f_1\}, \emptyset)$.

Note that the FDEP gate can trigger the failure of any gate (representing a subsystem) and not only BE as originally defined in Galileo [25]. Indeed, this extension comes at no extra cost, and the I/O-IMC used in this case is still the same as the one shown in Fig. 10a. Fig. 10b shows such a configuration, where T triggers the failure of the subsystem A. Note that subsystem A does not need to be an independent module. Note also that the trigger T only affects the failure of the gate A and none of its elements below it such as the basic event C.



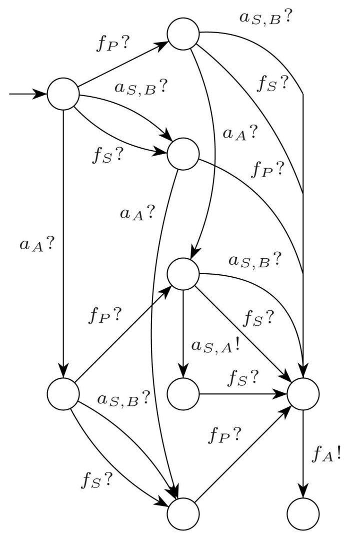


Fig. 11. The semantics $[\![(SPARE, 2)]\!]_{\mathrm{ELT}}(f_A, a_A, f_P, (f_S, a_{S,A}, \{a_{S,B}\}))$ of (left) SPARE gate.

When $l(v)$,[7] an element of the DFT, is triggered by multiple FDEP gates, then we define $T_v = \{f_t \mid \exists w \in V.(v, w) \in E' \wedge type(w) = FDEP \wedge t = (in'(w))_1\}$ as the set of trigger signals of FDEP gates on which $l(v)$ is dependent.

SPARE gate I/O-IMC model. Given the discussion in Section 2.2, Fig. 11 shows the I/O-IMC of a SPARE gate (the spare gate on the left side) sharing a spare with another SPARE gate. When the SPARE gate is active, the state reached after the primary fails is of particular interest. In this state, a nondeterministic situation arises where the spare can be activated by either of the SPARE gates (signals $a_{S,A}!$ and $a_{S,B}?$). This matches exactly the nondeterministic choice described in Section 2.1.

The semantics of a SPARE gate having $n - 1$ spares is a function $A^3 \times (A^2 \times \mathcal{P}(A))^{n-1} \rightarrow$ IOIMC that takes as inputs the firing signal and the activation signal of the SPARE gate, the firing signal of its primary, and a sequence of spare tuples containing, for each spare, its firing signal, its activation signal (output by the SPARE gate in question), and a list of spare activation signals of the other SPARE gates sharing that spare.

We now look at the IOIML definition of a spare gate with $n - 1$ (possibly shared) spares $[\![(SPARE, n)]\!]_{\mathrm{ELT}}(f_1, a_1, f_2, S)$, where f_1 is the failure signal of the spare gate, a_1 is the activation signal of the spare gate, f_2 is the failure signal of the primary component, $S = (f_3, a_{3,1}, P_3), \ldots, (f_{n+1}, a_{n+1,1}, P_{n+1})$, and $P_i = \{a_{i,2}, \ldots, a_{i,m}\}$. The set P_i is, in fact, the set of all activation signals of the ith spare by other spare gates. The signals $a_{x,y}$ then correspond to the activation of spare x by spare gate y. We separate the state space of the spare gate into four distinct sets as follows:

- DO. The spare gate is dormant and its primary is operational.
- DN. The spare gate is dormant and its primary is not operational.
- AO. The spare gate is active and its primary is operational.

7. Does not apply to FDEP gates.

- AN. The spare gate is active and its primary is not operational.

We now define the functions DO, DN, AO, AN:

$$DO(f, a, fp, S) = a?.AO(f, a, fp, S) + fp?.DN(f, a, fp, S) + \sum_{(k,l,M) \in S} \sum_{x \in \{k\} \cup M} x?.DO(f, a, fp, S - (k, l, M)),$$

$$DN(f, a, fp, S) = f!.\mathbf{0} \qquad \text{if } S = \emptyset,$$
$$DN(f, a, fp, S) = a?.AN(f, a, fp, S) + \sum_{(k,l,M) \in S} \sum_{x \in \{k\} \cup M} x?.DN(f, a, fp, S - (k, l, M)) \qquad \text{if } S \neq \emptyset,$$

$$AO(f, a, fp, S) = fp?.AN(f, a, fp, S) + \sum_{(k,l,M) \in S} \sum_{x \in \{k\} \cup M} x?.AO(f, a, fp, S - (k, l, M)),$$

$$AN(f, a, fp, S) = f!.\mathbf{0} \qquad \text{if } S = \emptyset,$$
$$AN(f, a, fp, S) = q!.AO(f, a, p, S - (p, q, R)) + \sum_{(k,l,M) \in S} \sum_{x \in \{k\} \cup M} x?.AN(f, a, fp, S - (k, l, M)) \qquad \text{if } S = (p, q, R), \ldots.$$

Now the IOIML definition of

$$[\![(SPARE, n)]\!]_{\mathrm{ELT}}(f_1, a_1, f_2, S) = DO(f_1, a_1, f_2, S).$$

The action signature of $[\![(SPARE, n)]\!]_{\mathrm{ELT}}(f_1, a_1, f_2, S)$ is:

$$(\{a_1, f_2\} \cup \{\{k\} \cup M \mid (k, l, M) \in S\}, \{f_1\} \cup \{l \mid (k, l, M) \in S\}, \emptyset).$$

The activation auxiliary. BEs and SPARE gates have a distinct input activation signal. When more than one SPARE gate can activate any of these two elements, it becomes convenient to carry out this activation through an intermediate I/O-IMC model called *activation auxiliary*.

Activating a BE (or a SPARE gate), $l(v)$ is done by composing $[\![l(v)]\!]_{\mathrm{ELT}}$ in parallel with an activation auxiliary I/O-IMC model, where the latter outputs the activation signal a_v of $l(v)$. The activation auxiliary I/O-IMC model is obtained through a function $AA : A \times \mathcal{P}(A) \rightarrow$ IOIMC that takes as arguments an output activation signal and a set of input activation signals (emitted by some SPARE gates). The activation auxiliary behaves similarly to an OR gate: It outputs the activation signal as soon as it receives an activation signal emitted by one of the SPARE gates.

The general form of v's activation auxiliary function AA is $AA(a_v, Atv_v)$, where a_v is v's activation signal and

$$\begin{aligned} Atv_v = \{a_{w,sp} \mid \mathit{type}(sp) = SPARE \wedge w &\in (in'(sp) \setminus (in'(sp))_1) \wedge v \in stb(w) \wedge (\nexists\, v_0, v_1 \ldots v_n \\ &\in V, n \geq 0.v_0 = v, v_n = w \wedge \forall 0 \leq i < n.(v_i, v_{i+1}) \\ &\in E'' \wedge \mathit{type}(v_{i+1}) = SPARE \wedge v_i \\ &\in (in'(v_{i+1}) \setminus (in'(v_{i+1}))_1))\} \end{aligned}$$

is the set of activation signals emitted by all SPARE gates sharing v. The last clause simply ensures that there is no directed path from v to w containing an edge that is a spare input (i.e., nonprimary) to a SPARE gate. It is important to note that activation does not propagate through an FDEP-dependent event input.

Thus, we can write

$$AA(a_v, Atv_v) = [\![(OR, n)]\!]_{\mathrm{ELT}}(a_v, (Atv_v)_1, \ldots, (Atv_v)_n),$$

given that $|Atv_v| = n$ $(n > 0)$. The action signature of $AA(a_v, Atv_v)$ is $(Atv_v, a_v, \emptyset)$. If $n = 0$ (i.e., no explicit activation by a SPARE gate, and therefore, activated when system starts at time $t = 0$), then $AA(a_v, \emptyset) = a_v!.\mathbf{0}$.

Complete semantics of a DFT. To obtain the semantics of a DFT from the semantics of its elements, we need to appropriately instantiate the parameters of $[\![l(v)]\!]_{\mathrm{ELT}}$ (we sometimes use $[\![v]\!]_{\mathrm{ELT}}$for short) of each node v. We use the following notations: 1) The firing signal f_v of element $l(v) \in \mathcal{E}$ denotes the failure of v, 2) the activation signal a_v denotes its[8] activation, and 3) $a_{v,u}$ denotes the activation signal output by a SPARE gate u to activate spare v. We also introduce the following notations: V_{BE} is the set of all nodes $v \in V$ s.t. $\mathit{type}(v) = BE$ or $\mathit{type}(v) = PHBE$, V_{AOVP} is the set of all nodes $v \in V$ s.t.

$$\mathit{type}(v) = AND \vee OR \vee VOT \vee PAND,$$

and V_{SPARE} is the set of all nodes $v \in V$ s.t. $\mathit{type}(v) = SPARE$. Now, the semantics of a DFT is obtained by parallel composing the semantics of all (non-FDEP) nodes.

Definition 10. *The semantics of a DFT* $\mathcal{D} = (V, in, l)$ *is the I/O-IMC*

$$\begin{aligned} [\![\mathcal{D}]\!] = &\, \|_{v \in V_{BE}} [\![v]\!]_{\mathrm{ELT}}(a_v, f_v^*) \| FA(f_v, f_v^*, T_v) \| AA(a_v, Atv_v) \\ &\|_{v \in V_{AOVP}} [\![v]\!]_{\mathrm{ELT}}(f_v^*, f_{w_1}, f_{w_2}, \ldots f_{w_n}) \| FA(f_v, f_v^*, T_v) \\ &\|_{v \in V_{SPARE}} [\![v]\!]_{\mathrm{ELT}}(f_v^*, a_v, f_{w_1}, S_2, \ldots, S_n) \| FA(f_v, f_v^*, T_v) \| \\ &AA(a_v, Atv_v), \end{aligned}$$

where $in'(v) = w_1 w_2 \ldots w_n$ and $S_i = (f_{w_i}, a_{w_i,v}, P_{w_i})$ with $i > 1$ is a tuple which gives, for spare $l(w_i)$ the failure signal (f_{w_i}), the activation signal by SPARE gate $l(v)$ $(a_{w_i,v})$, and the set of activation signals emitted by all SPARE gates (except v) sharing spare $l(w_i)$:

$$P_{w_i} = \{a_{w_i,g} \mid (w_i, g) \in E' \wedge g \neq v \wedge \mathit{type}(g) = SPARE\}.$$

To compute the reliability of $\mathcal{D}$, we are only interested in the failure of the top node T. Hence, we hide all signals except f_T, i.e., we compute $M_{\mathcal{D}} = \mathit{hide}\ A_{\mathcal{D}} \setminus f_T\ \mathit{in}\ [\![\mathcal{D}]\!]$; recall that $A_{\mathcal{D}}$ denotes the set of all actions in $\mathcal{D}$. The compositional aggregation technique described in Section 7 is an efficient way to derive $M_{\mathcal{D}}$.

Example 2. Fig. 12 shows the I/O-IMC semantics of a DFT consisting of a SPARE gate A having a primary B and a spare C. Since the DFT contains no FDEP gates, we

8. Only for BEs and SPARE gates.

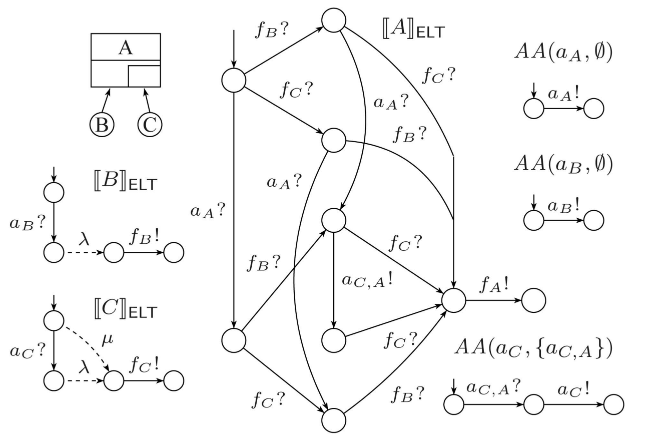


Fig. 12. A DFT example and the six I/O-IMCs that model its behavior.

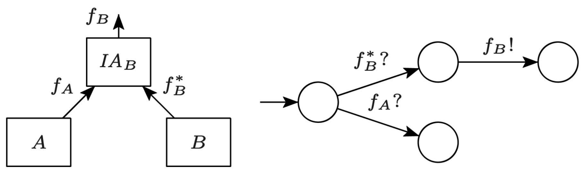


Fig. 13. The I/O-IMC model of the IA.

ignore all firing auxiliaries. The I/O-IMC of the DFT is obtained by parallel composing $[\![A]\!]$, $[\![B]\!]$, and $[\![C]\!]$:

$$[\![A]\!] = [\![(SPARE, 2)]\!]_{\mathrm{ELT}}(f_A, a_A, f_B, (f_C, a_{C,A}, \emptyset))\|AA(a_A, \emptyset),$$
$$[\![B]\!] = [\![(BE, 0, \lambda, 0)]\!]_{\mathrm{ELT}}(a_B, f_B)\|AA(a_B, \emptyset),$$
$$[\![C]\!] = [\![(BE, 0, \lambda, \mu)]\!]_{\mathrm{ELT}}(a_C, f_C)\|AA(a_C, \{a_{C,A}\}).$$

6 DFT Elements Extension

In this section, we show, through three examples, how one can readily extend the DFT elements within the I/O-IMC framework. These extensions concern the modeling of *inhibition*, *mutually exclusive events*, and *repair*.

Adding/modifying elements is done at the level of the elementary I/O-IMC models. Moreover, adding/modifying one element does not affect the remainder of the elements (i.e., their corresponding I/O-IMC models). This is indeed a desirable property of the I/O-IMC framework, where the behavioral details and interactions of any element are kept as local as possible. These extensions only affect Step 1 of the DFT conversion/analysis algorithm laid out in Section 7, leaving the other five steps unchanged.

Inhibition and mutual exclusion. We say that event A *inhibits* the failure of B if the failure of B is prevented when A fails before B. Following the idea of the firing auxiliary (cf., Section 5), this could be modeled by simply adding an *inhibition auxiliary* (IA). Fig. 13 shows the configuration of such inhibition and the corresponding I/O-IMC model of the IA of B. f^*_B corresponds to the failure signal of B taken in isolation, i.e., without A's inhibition. Note that, as with the FA, any element which has B as input has to now interface with B's IA rather than directly with B.

If A inhibits the failure of B and B also inhibits the failure of A, the failure of A and the failure of B become two

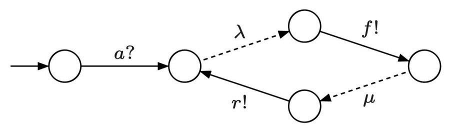


Fig. 14. The repairable BE I/O-IMC model.

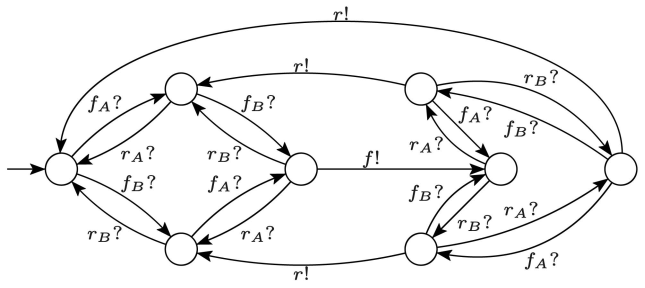


Fig. 15. The repairable AND gate I/O-IMC model.

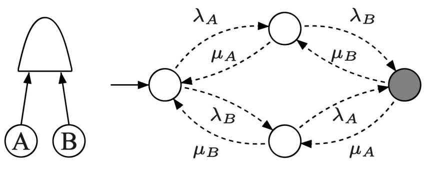


Fig. 16. A simple repairable system. The gray state denotes the state in which the DFT has failed.

mutually exclusive events. Clearly, this can be modeled in our framework by adding IAs for both A and B. Mutual exclusion is very useful for modeling different failures of a single component with different effects (e.g., a valve being stuck open or closed).

Repair. Adding a notion of repair is somewhat more complicated as every DFT element can now fail or be repaired. Thus, no longer only a "failed event" should be signaled but also a "repaired event." However, as mentioned above, we only need to modify "locally" the elementary I/O-IMC corresponding to each DFT element behavior. Here, we will only discuss the new I/O-IMC for the BE and the AND gate (other elements are treated in the same fashion). The repairable cold BE's I/O-IMC is shown in Fig. 14. Here, μ denotes the BE repair rate and $r!$ is a signal output by the BE notifying the rest of the elements that it has been repaired. The repairable AND gate I/O-IMC model is shown in Fig. 15. The AND gate has its own repair output signal (i.e., $r!$) and needs to consider both failure ($f_A?$ and $f_B?$) and repair ($r_A?$ and $r_B?$) signals coming from its inputs A and B. Compared to the unrepairable AND gate (Fig. 9), Fig. 15 has three extra states. If we consider a very simple repairable system composed of an AND gate with two BEs A and B (Fig. 16a), then the resulting I/O-IMC after automatic composition, hiding of all signals and aggregation is, as expected, the CTMC shown in Fig. 16b. At this point, one can perform analysis on the CTMC such as computing the system unavailability.

7 Compositional Aggregation Approach

The technique of *compositional aggregation* consists of composing a large model out of smaller ones and aggregating submodels after each compositional step. This approach is to be contrasted with a more classical approach of model generation, such as the one used by Galileo DIFTree [21], where the model of a system is generated at once and as a whole and then eventually aggregated at the end. Compositional aggregation is very effective in combating the state-space explosion problem and has been already successfully used on a number of case studies, most notably in [18].

Once the DFT elements have been converted into a set of I/O-IMCs, the compositional aggregation methodology can be applied to combine the set into a single I/O-IMC. The final I/O-IMC reduces in many cases to a CTMC. This CTMC can then be solved using standard methods [24] to compute performance measures such as system unreliability. The conversion/analysis algorithm[9] is as follows:

1. Map each DFT element to its corresponding (aggregated) I/O-IMC and match all inputs and outputs. The result of this step is a set of I/O-IMCs.
2. Pick two I/O-IMCs and parallel compose them.
3. Hide output signals that won't be subsequently used (i.e., synchronized on).
4. Aggregate (using weak bisimulation) the I/O-IMC obtained from the composition of the two I/O-IMCs picked in Step 2 and the hiding of the output signals in Step 3.
5. Go to Step 2 if more than one I/O-IMC is left; otherwise, go to Step 6.
6. Analyze the aggregated CTMC.

The choice of I/O-IMCs made in step 2 is important as it influences the size of the generated state space during the intermediate steps. If no nondeterminism is present in the DFT model, then the algorithm yields a CTMC. However, in some cases where nondeterminism arises, the result is a continuous-time Markov decision process, which can be analyzed by computing bounds on the performance measure of interest [2].

8 Tool Support

In this section, we describe the CORAL [6] tool chain, which supports our DFT analysis methodology. The tools presented in this section use the CADP tool set [15] for many operations on I/O-IMCs, such as composition, aggregation, and CTMC analysis. Before discussing the various tools in detail, we will first give an overview of the tool chain.

Tool chain overview. Fig. 17 shows an overview of the tool chain for our DFT analysis methodology. The user must supply the following inputs: the DFT in a file using the Galileo format, a composition script denoting the order of composition used in the compositional aggregation phase, and the mission times for the unreliability analysis.

A DFT can be analyzed by performing the following three steps, which are elaborated below:

1. call the *dft2bcg* tool with as input the DFT file in extended Galileo format,[10]
2. call the *composer* tool with as input a composition script, and

9. Note that this algorithm is amenable to parallelization.
10. The standard Galileo format is extended to allow complex spares and dependent events.

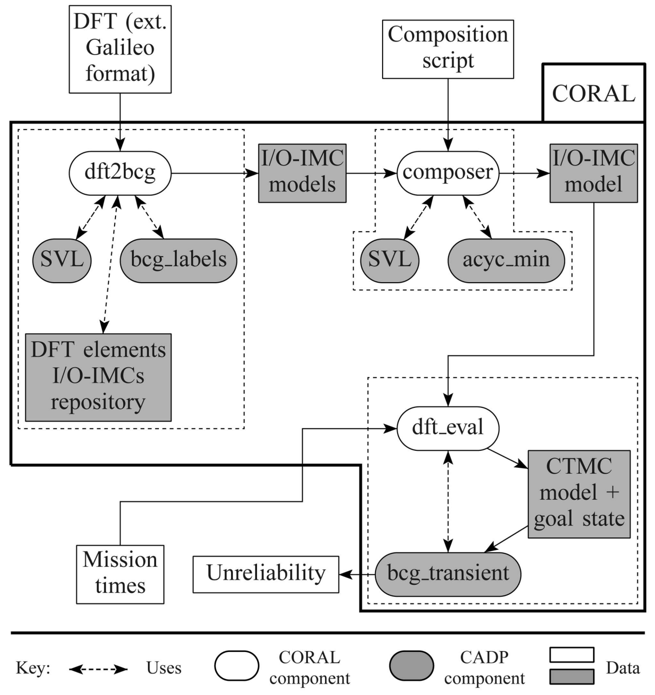


Fig. 17. An overview of the CORAL tool chain.

3. call the *dft_eval* tool with as input a number of mission times. The dft_eval tool then calculates the unreliability of the system modeled by the original DFT for the given mission times. It generates a CTMC model describing the exact failure distribution of the system if there is no nondeterminism.

Generating I/O-IMC models: dft2bcg. The dft2bcg tool generates a number of I/O-IMC models that describe the behavior of a given DFT. To be more exact, each of the I/O-IMCs describes one element in the DFT (see Section 5). These I/O-IMC models are stored in binary coded graph (BCG) format supported by CADP. The dft2bcg tool uses a number of script verification language (SVL) scripts to generate BCG files. These scripts are interpreted by the SVL tool, which is part of the CADP tool set, to perform generation, parallel composition, hiding, and minimization of BCG files. dft2bcg also uses the bcg_labels tool of the CADP tool set, which allows the renaming of the actions of an I/O-IMC.

The dft2bcg tool performs the following steps to generate the I/O-IMC models:

1. parses the DFT input file,
2. checks the validity of the DFT (i.e., syntactic check),
3. calls the SVL tool to generate I/O-IMC models with generic action names ($f_1, f_2, \dots$) using the DFT SVL scripts, and
4. calls the bcg_labels tool to create the I/O-IMC models by renaming the generic actions to the specific actions derived from the names of the DFT elements.

All generic models generated in step 3 are stored in a DFT repository for reuse in later calls of the dft2bcg tool. For instance, if we need three different 3-input AND-gates, we can simply use the same generic 3-input AND gate, renaming it differently in step 4 of the dft2bcg tool. The

TABLE 1
Results of the Case Studies

Case study	Approach	Max # of states	Max # of transitions	Unreliability (time = 1)	Time (sec)
CPS	Galileo	4113	24608	0.00135	490
	CORAL	133	465	0.00135	67
CAS	Galileo	8	10	0.65790	1
	CORAL	36	119	0.65790	94
CAS-PH	*CORAL*	40052	265442	0.112826	231
FTPP-4	Galileo	32757	426826	0.01922	13111
	CORAL	1325	13642	0.01922	65
FTPP-5	*CORAL*	43105	643339	0.00306	309
FTPP-6	*CORAL*	1180565	22147378	0.000453	1989
FTPP-C	*CORAL*	653303	12220653	0.02136	1806
FTPP-A	Galileo	32757	426826	0.0167	13111
	CORAL	19367	154566	0.0167	240
NDPS	*CORAL*	61	169	[0.00586, 0.00598]	266

repository also holds a number of basic I/O-IMC models, which are used to generate the models of all DFT elements.

Compositional aggregation: composer. We have seen above that the dft2bcg tool generates a number of I/O-IMC models. The composer tool uses as input these I/O-IMC models and a composition script supplied by the user. The composition script describes the order in which the I/O-IMC models should be composed. The composer tool executes the commands in the script, composing the I/O-IMC models into a single I/O-IMC model that represents the stochastic behavior of the entire DFT. Our choice of composition script is based on heuristics, such as maximizing the number of transitions that will be hidden and minimizing the number of actions that are not synchronized. After each composition, the resulting I/O-IMC model is minimized using the acyc_min tool [12]. The acyc_min tool minimizes acyclic (except for self-loops with input actions) I/O-IMCs with respect to weak bisimulation for I/O-IMCs (see Section 3).

Calculating measures: dft_eval. In many cases, the stochastic behavior of the system described by a DFT can be modeled as a CTMC. To be more specific, if there is no nondeterminism present in the DFT model of the system, the I/O-IMC generated in our approach reduces to a CTMC. See Section 2.1 for a detailed discussion on the occurrence of nondeterminism in DFT analysis. The dft_eval tool first reduces the I/O-IMC representation of the DFT into a CTMC and then invokes the CADP tool bcg_transient to find the unreliability of the DFT for a set of mission times supplied by the user.

9 Case Studies

We have assessed the efficiency of our compositional aggregation approach by performing nine case studies from different application areas. We analyzed a cascaded PAND system (CPS), two versions of a cardiac assist system (CAS), five versions of a fault-tolerant parallel processors (FTPPs), and finally, a pump system with inherent nondeterminism. We systematically compare our results (using the *CORAL* tool) to the Galileo DIFTree tool [14] results, see Table 1 for an overview. Here, the number of states/transitions corresponds to the largest I/O-IMC or CTMC encountered during analysis. All experiments were run on an AMD Athlon XP 2,600+ running at 1.9 GHz with 1 GB memory.

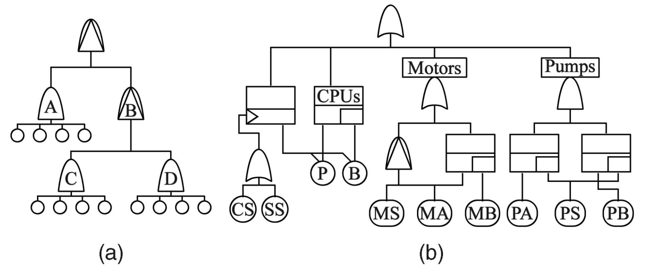


Fig. 18. DFTs for CPS (a) and CAS (b) case studies.

The cascaded PAND system. This system, taken from [7] and shown in Fig. 18a, illustrates the enhanced modularity of our methodology compared to Galileo DIFTree. In fact, given that the top node of the tree is a PAND dynamic gate, Galileo DIFTree can only consider the tree as a whole when generating/solving its corresponding CTMC. In our compositional aggregation approach, we realize that there are independent modules, in particular, A, C, and D are all identical (all BEs have a failure rate equal to 1) and independent modules. In fact, it suffices to generate and aggregate the I/O-IMC of one of these three modules and reuse the result, given some renaming of signals, for the remaining two modules. In this way, A, C, and D each have an aggregated I/O-IMC of seven states only. The CTMC generated by Galileo has 4,113 states. This result is to be compared with the largest I/O-IMC of 113 states obtained during the compositional aggregation.

The cardiac assist system. This system, taken from [5] and shown in Fig. 18b, consists of three separate modules (i.e., CPU, motors, and pump units). Table 2 shows the failure rates of the various components. In addition, B is a warm spare with a dormancy factor $\alpha = 0.5$, and MB and PS are cold spares (i.e., $\alpha = 0$). During analysis, Galileo DIFTree modularizes the DFT into three independent modules (namely CPU, motors, and pump units) and generates a separate CTMC for each one of them. The biggest CTMC has eight states. The CTMCs' results are then combined (through the top OR gate) using BDDs. Using the compositional aggregation approach, and without modularization, the biggest I/O-IMC encountered has 36 states. The results of CAS are summarized in Table 1. Here, clearly Galileo outperforms *CORAL* because it uses modularization which has not been implemented yet in *CORAL*. If we switch off modularization in Galileo (i.e., generate a single CTMC for the whole system), then it produces a CTMC with 85 states.

To illustrate the possibility of using phase-type distributions, we have modified the CAS case study by replacing BEs with PHBEs (case CAS-PH). In this case, all basic events occur after a delay governed by an Erlang distribution with four phases and the same expectation as the exponential

TABLE 2
Failure Rates for CAS

Component	CS	SS	P	B	MA	MB	MS	PA	PB	PS
Rate	0.2	0.2	0.5	0.5	1	1	0.01	1	1	1

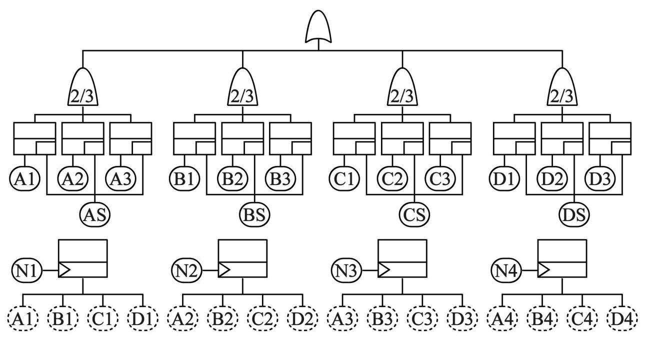

2/3
2/3
2/3
2/3
A1 A2 A3 B1 B2 B3 C1 C2 C3 D1 D2 D3
AS BS CS DS
N1 N2 N3 N4
A1 B1 C1 D1 A2 B2 C2 D2 A3 B3 C3 D3 A4 B4 C4 D4

Fig. 19. DFT for the FTPP-4 case study.

delay from the CAS case study (for instance, PHBE CS is governed by a 4-phase Erlang with rate parameter 0.8 instead of an exponential distribution with rate 0.2). For the passive delay of warm spare B, we also use an Erlang distribution with four phases and the same expectation as the passive exponential delay.

The fault-tolerant parallel processors. This system, taken from [5], consists of 16 processors divided into 4 logical groups. In each group, a processor is used as a shared cold spare. A network element (NE) physically connects 1 processor in each group (thus there are 4 NEs) to the rest of the system. The failure of an NE makes the four processors connected to it unavailable (i.e., essentially failed). The requirement is to have at least two processors operational in each group. The DFT is shown in Fig. 19, where the processors are denoted by A_i, B_i, C_i, D_i, and the network elements with N. All network elements have a failure rate equal to 0.017, and all processors have a failure rate equals 0.11. The four spare processors are cold spares (i.e., $\alpha = 0$).

To illustrate even further the state-space explosion problem, we took the FTPP system and made it larger by considering 5 (FTPP-5), respectively, 6 (FTPP-6) processors in each group. For these case studies, the Galileo tool runs out of memory. The (FTPP-C) case study is an extension to the FTPP system, where each of the processors (e.g., A_1) is replaced by a complex element that consists of an OR-gate with two inputs: a BE “CPU” and a spare gate “memory” with primary BE “M1” and cold spare BE “M2.” All these BEs have failure rate 0.11. No results are available from Galileo DIFTree as this kind of extended DFTs cannot be handled by Galileo. It is interesting to investigate how dependent the success of the compositional aggregation technique is on the symmetries in the DFT models. In the case study FTPP-A, we have therefore considered a variant of the FTPP case study, where almost all basic events have different rates. The rates used are given in Table 3. We see that indeed the models encountered are larger by a factor of

TABLE 3
Failure Rates for FTPP-A

BE	Rate	BE	Rate	BE	Rate	BE	Rate	BE	Rate
N_1	3.4	A_1	5.5	B_1	11	C_1	6.6	D_1	7.7
N_2	2.9	A_2	7.7	B_2	15.4	C_2	99	D_2	11
N_3	2.4	A_3	14	B_3	28	C_3	16	D_3	22
N_4	1.7	A_4	18	B_4	36	C_4	22	D_4	30

Rates are scaled up by a factor of 10^5.

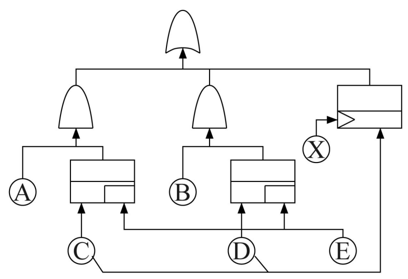


Fig. 20. DFT for NDPS case study.

15, but still the computational time required is manageable at under five minutes.

Nondeterministic pump system (NDPS). This case study illustrates how we deal with nondeterminism in DFT models. The I/O-IMC model of the DFT is generated as usual and then converted into a CTMDP following [17]. This CTMDP is then analyzed using the MRMC [19] tool to obtain maximum and minimum unreliability. The DFT is a simple pumping system with five pumps: main pumps A and B, auxiliary pumps C, D, and spare auxiliary pump E (see Fig. 20). BE A has rate 0.05 and BE B has rate 0.1; BEs C and D both have rate 0.2 and BE E has rate 0.3 with dormancy $\alpha = 0$. Finally, BE X has rate 0.002. This DFT is nondeterministic since C and D may fail at the same time due to their dependence on BE X. It is then undetermined whether spare E replaces C or D. We can see that the maximum and minimum unreliabilities are not far apart, this is caused by the low probability of the “problematic” BE X.

10 Conclusion and Future Work

In this paper, we have formalized the syntax and semantics of DFTs and introduced a DFT analysis framework based on I/O-IMCs, increasing the DFT modularity both at the analysis level and the model building level. We have also demonstrated the ease with which one can define new DFT elements and provided examples of such extensions. Finally, we have built a prototype tool for analyzing DFTs named CORAL and run some experiments to compare our methodology and results to the Galileo DFT tool.

Areas of future research include: 1) From a process algebra point of view, we would like to achieve even more drastic state-space reduction using more suitable aggregation techniques. 2) Adapt the I/O-IMC approach to other formalisms, such as the Architecture Analysis and Design Language [1]. A first step toward this goal has been made in [4].

Acknowledgments

The authors thank Holger Hermanns and the anonymous reviewers of this paper for their helpful comments. This research has been partially funded by the Netherlands Organization for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.000.505 (MOQS); by the EU under grant IST-004527 (ARTIST2) and FP7-ICT-2007-1 grant 214755 (QUASIMODO); and by the DFG/NWO bilateral cooperation program under project number DN 62-600 (VOSS2).

References

[1] As-2 Embedded Computing Systems Committee, Architecture Analysis & Design Language (AADL), Number: AS5506, Revision: A, Jan. 2009.

[2] C. Baier, H. Hermanns, J.P. Katoen, and B.R. Haverkort, "Efficient Computation of Time-Bounded Reachability Probabilities in Uniform Continuous-Time Markov Decision Processes," *Theoretical Computer Science,* vol. 345, no. 1, pp. 2-26, 2005.

[3] E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, J. Rakow, R. Wimmer, and B. Becker, "Compositional Dependability Evaluation for Statemate," *IEEE Trans. Software Eng.*, vol. 35, no. 2, pp. 274-292, Mar./Apr. 2009.

[4] H. Boudali, P. Crouzen, B.R. Haverkort, M. Kuntz, and M. Stoelinga, "Architectural Dependability Evaluation with Arcade," *Proc. 38th IEEE/IFIP Int'l Conf. Dependable Systems and Networks,* pp. 512-521, 2008.

[5] H. Boudali, P. Crouzen, and M. Stoelinga, "A Compositional Semantics for Dynamic Fault Trees in Terms of Interactive Markov Chains," *Proc. Fifth Int'l Symp. Automated Technology for Verification and Analysis,* pp. 441-456, 2007.

[6] H. Boudali, P. Crouzen, and M. Stoelinga, "Coral—a Tool for Compositional Reliability and Availability Analysis," *Proc. ARTIST WS: Tool Platforms for ES Modelling, Analysis and Validation,* 2007.

[7] H. Boudali, P. Crouzen, and M. Stoelinga, "Dynamic Fault Tree Analysis Using Input/Output Interactive Markov Chains," *Proc. 37th IEEE/IFIP Int'l Conf. Dependable Systems and Networks,* pp. 708-717, June 2007.

[8] M.A. Boyd, "Dynamic Fault Tree Models: Techniques for Analyses of Advanced Fault Tolerant Computer Systems," PhD dissertation, Dept. of Computer Science, Duke Univ., 1991.

[9] M. Bravetti and R. Gorrieri, "The Theory of Interactive Generalized Semi-Markov Processes," *Theoretical Computer Science,* vol. 282, no. 1, pp. 5-32, 2002.

[10] D. Coppit, K.J. Sullivan, and J.B. Dugan, "Formal Semantics of Models for Computational Engineering: A Case Study on Dynamic Fault Trees," *Proc. Int'l Symp. Software Reliability Eng.,* pp. 270-282, Oct. 2000.

[11] N. Coste, H. Garavel, H. Hermanns, R. Hersemeule, Y. Thonnart, and M. Zidouni, "Quantitative Evaluation in Embedded System Design: Validation of Multiprocessor Multithreaded Architectures," *Proc. Conf. Design, Automation and Test in Europe,* pp. 88-89, 2008.

[12] P. Crouzen, H. Hermanns, and L. Zhang, "On the Minimisation of Acyclic Models," *Proc. 19th Int'l Conf. Concurrency Theory,* pp. 295-309, 2008.

[13] J.B. Dugan, S.J. Bavuso, and M.A. Boyd, "Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems," *IEEE Trans. Reliability,* vol. 41, no. 3, pp. 363-377, Sept. 1992.

[14] J.B. Dugan, B. Venkataraman, and R. Gulati, "DIFTree: A Software Package for the Analysis of Dynamic Fault Tree Models," *Proc. Reliability and Maintainability Symp.,* pp. 64-70, Jan. 1997.

[15] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, "Cadp 2006: A Toolbox for the Construction and Analysis of Distributed Processes," *Proc. 19th Int'l Conf. Computer Aided Verification (CAV),* 2006.

[16] H. Hermanns, *Interactive Markov Chains.* Springer Berlin, 2002.

[17] H. Hermanns and S. Johr, "Uniformity by Construction in the Analysis of Nondeterministic Stochastic Systems," *Proc. 37th IEEE/IFIP Int'l Conf. Dependable Systems and Networks,* pp. 718-728, 2007.

[18] H. Hermanns and J.P. Katoen, "Automated Compositional Markov Chain Generation for a Plain-Old Telephone System," *Science of Computer Programming,* vol. 36, no. 1, pp. 97-127, 2000.

[19] J.-P. Katoen, M. Khattri, and I.S. Zapreev, "A Markov Reward Model Checker," *Proc. Second Int'l Conf. Quantitative Evaluation of Systems,* pp. 243-244, 2005.

[20] N.A. Lynch and M.R. Tuttle, "An Introduction to Input/Output Automata," *CWI Quarterly,* vol. 2, no. 3, pp. 219-246, 1988.

[21] R. Manian, J.B. Dugan, D. Coppit, and K.J. Sullivan, "Combining Various Solution Techniques for Dynamic Fault Tree Analysis of Computer Systems," *Proc. IEEE Int'l High-Assurance Systems Eng. Symp.,* vol. 3, pp. 21-28, 1998.

[22] M.F. Neuts, *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach.* Dover, 1981.

[23] Relex "Fault Tree and Event Tree Software," http://www.relex.com/products/ftaeta.asp, 2008.

[24] W.J. Stewart, *Introduction to the Numerical Solution of Markov Chains.* Princeton Univ. Press, 1994.

[25] K.J. Sullivan, J.B. Dugan, and D. Coppit, "The Galileo Fault Tree Analysis Tool," *Proc. 29th Ann. Int'l Symp. Fault-Tolerant Computing,* pp. 232-235, June 1999.

[26] K.K. Vemuri, J.B. Dugan, and K.J. Sullivan, "Automatic Synthesis of Fault Trees for Computer-Based Systems," *IEEE Trans. Reliability,* vol. 48, no. 4, pp. 394-402, Dec. 1999.

[27] W.E. Veseley, F.F. Goldberg, N.H. Roberts, and D.F. Haasl, technical report, Fault Tree Handbook, NUREG-0492., NASA, 1981.

Hichem Boudali received the computer science engineering degree (MSc equivalent) from the Polytechnic School and Applied Sciences at the Université Libre de Bruxelles, Brussels, Belgium, in 2002, the MSc degree in electrical engineering from the University of Virginia, and the PhD degree in computer engineering from the School of Engineering and Applied Science at the University of Virginia, Charlottesville, in 2005. He is a member of the IEEE.

Pepijn Crouzen received the computer science engineering degree (MSc equivalent) from the University of Twente, Enschede, Netherlands, in 2006. He is now working toward the PhD degree in the Dependable Systems and Software Group of the Computer Science Faculty at Saarland University, Saarbrücken, Germany.

Mariëlle Stoelinga received the PhD degree in computer science from the University of Nijmegen, Netherlands, in 2002. She is currently an assistant professor in the Formal Methods and Tools Group at the University of Twente, Netherlands.