Dynamic Fault Tree analysis using Input/Output Interactive Markov Chains[*]

Hichem Boudali
hboudali@cs.utwente.nl

Pepijn Crouzen[§,†]
crouzen@alan.cs.uni-sb.de

Mariëlle Stoelinga
marielle@cs.utwente.nl

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

[§]Saarland University, Department of Computer Science,
D-66123 Saarbrücken, Germany.

Abstract

Dynamic Fault Trees (DFT) extend standard fault trees by allowing the modeling of complex system components' behaviors and interactions. Being a high level model and easy to use, DFT are experiencing a growing success among reliability engineers. Unfortunately, a number of issues still remains when using DFT. Briefly, these issues are (1) a lack of formality (syntax and semantics), (2) limitations in modular analysis and thus vulnerability to the state-space explosion problem, and (3) lack in modular model-building. We use the input/output interactive Markov chain (I/O-IMC) formalism to analyse DFT. I/O-IMC have a precise semantics and are an extension of continuous-time Markov chains with input and output actions. In this paper, using the I/O-IMC framework, we address and resolve issues (2) and (3) mentioned above. We also show, through some examples, how one can readily extend the DFT modeling capabilities using the I/O-IMC framework.

KEYWORDS: Fault tree, Interactive process, Markov chain, compositional aggregation, modularity.

1. Introduction

Dynamic fault trees (DFT) [10, 7, 19] extend standard (or static) fault trees (FT) [20] by defining additional gates called dynamic gates. These gates allow the modeling of complex system components' behaviors and interactions which is far superior to the modeling capabilities of standard FT. Like standard FT, dynamic fault trees are a high-level formalism for computing reliability measures of computer-based systems, such as the probability that the system fails during its mission time. For over a decade now, DFT have been experiencing a growing success among reliability engineers. Unfortunately, a number of issues still remains when using DFT. Most notably the following three issues are a matter of concern: (1) 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. (2) DFT lack modular analysis. That is, even though stochastically-independent sub-modules exist in a certain DFT module (specifically those whose top-node is a dynamic gate), these sub-modules cannot be solved separately and still get an exact solution. Consequently, a DFT model, which is typically analyzed by first converting it into a Markov chain (MC), becomes vulnerable to the state space explosion problem. (3) DFT also lack modular model-building, i.e. there are some rather severe restrictions on the type of allowed inputs to certain gates (e.g. inputs to spare gates and dependent events of functional dependency gates have to be basic events), which greatly diminish the modeling flexibility and power of DFT.

DFT are comprised of various elements[1]: Basic events, static gates (AND, OR, and K/M gates), and dynamic gates (functional dependency, priority AND, and spare gates). Each of these elements is viewed as a process moving from one state to another. States denote either the operation or the failure of the element. Each element, or process, also interacts (communicates) with its environment by responding to certain input signals and producing output signals. These elements[2] could also possess a purely stochastic behavior by allowing (in a probabilistic fashion) the passage of time prior to moving to another state. In the remainder

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

[†]The majority of this work was done while the author was at the University of Twente.

[1]Also called components.

[2]At this point only the basic events.

of the paper, we assume this passage of time to be governed by an exponential probability distribution (thus behaving as a Markovian process). Moving from one state to another is therefore caused by either an input or output transition or due to a Markovian transition.

Given the nature of DFT elements we have used the input/output interactive Markov chain (I/O-IMC) formalism [4] to model the semantics of DFT. In fact, I/O-IMC augment continuous-time Markov chains with input and output actions and a clear separation between Markovian transitions and interactive (involving input or output actions) transitions is made. Furthermore, I/O-IMC have a precise and formal semantics and have proved to be a suitable and natural way to model DFT elements.

I/O-IMC are an example of a stochastic extension to a process algebra. These stochastic process algebras have recently gained popularity in performance modeling and analysis due to their *compositional aggregation* approach. We refer the reader to [14] for case studies on the application of the compositional aggregation approach to the modeling and analysis of real systems. Compositional aggregation is a technique to build an I/O-IMC by composing, in successive iterations, a number of elementary and smaller I/O-IMC and reducing (i.e. aggregating) the state-space of the generated I/O-IMC as the composition takes place (cf. Section 3).

Issue (1), mentioned above, has been addressed in [4] where a formal syntax and semantics for DFT have been defined. The formal syntax is derived by characterizing the DFT as a directed acyclic graph. The formal DFT semantics is described in terms of I/O-IMC, and provides a rigorous basis for the analysis of DFT. In fact, each DFT element has a corresponding elementary I/O-IMC. This semantics is fully compositional, that is, the semantics of a DFT is expressed in terms of the semantics of its elements. This enables an efficient analysis of DFT through compositional aggregation to produce a single I/O-IMC, on which we can then carry out performance analysis. Earlier work on formalizing DFT can be found in [8], where DFT are specified using the Z formal specification language. The main difference between the formal specification in [8] 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 allows us to use the well-defined concept of compositional aggregation which helps us to combat the state-space explosion problem. In fact, this notion of compositional aggregation is not present in [8] and the state-space explosion problem is not addressed or mitigated whatsoever.

We address issue (2) by showing, using the I/O-IMC framework, how the DFT analysis becomes greatly modular compared to current state of the art DFT analysis techniques. In particular, we demonstrate, through an example system, how an I/O-IMC corresponding to a certain (independent) dynamic module[3] can be reused in any larger DFT model.

We also tackle issue (3) and lift two previously enforced restrictions on DFT; namely, the restriction on spares and functional dependency gates' dependent events to be basic events. In fact, in our framework it becomes possible to, for instance, model a spare as a complex sub-system comprised of several basic events and gates. The use of (shared) spares in DFT has always been somehow problematic [8]. In this paper, we carefully examine, clarify, and generalize the concept of *spare activation*.

To summarize, we make the following contributions:

1. Illustrate, through a case study, the use of the I/O-IMC framework for the analysis of DFT, and in particular we show the benefits of the compositional aggregation approach.

2. Show the increased DFT modular analysis and the concept of reuse of dynamic modules.

3. Extend the DFT modeling capabilities by allowing complex spares (through the generalization of the concept of activation) and complex functionally dependent events.

4. Illustrate how readily one can define new DFT elements and provide 3 examples.

The remainder of the paper is organized as follows: In Section 2 and Section 3, we introduce DFT and I/O-IMC respectively. In Section 4, we show how a DFT is automatically converted into a community of I/O-IMC and discuss non-determinism. In Section 5, we illustrate the DFT modular analysis. In Section 6, we lift the restrictions on the spare and functional dependency (FDEP) gates. Finally, in Section 7, we illustrate how one can readily extend the modeling capabilities of DFT by augmenting or modifying the set of elementary I/O-IMC models. Some of these extensions include *mutually exclusive events* and *repair*. We conclude the paper and suggest future work in Section 8.

2. Dynamic fault trees

A fault tree model describes the system failure in terms of the failure of its components. Standard FT are combinatorial models and are built using static gates (the AND, the OR, and the K/M gates) and basic events (BE). A combinatorial model only captures the combination of events and not the order of their occurrence. Combinatorial models become, therefore, inadequate to model today's complex dynamic systems. DFT introduce three novel modeling capabilities: (1) spare component management and allocation,

[3]Also called sub-system or sub-tree.

(2) functional dependency, and (3) failure sequence dependency. These modeling capabilities are realized using three main dynamic gates[4]: The spare gate, the functional dependency (FDEP) gate, and the priority AND (PAND) gate. Figure 1 depicts the three dynamic gates.

The PAND gate fails when all its inputs fail and fail from left to right (as depicted on the figure) order. The spare gate has one primary input and one or more alternate inputs (i.e. the spares). The primary input is initially powered on and when it fails, it is replaced by an alternate input. The spare gate fails when the primary and all the alternate inputs fail (or are unavailable). A spare could also be shared among multiple spare gates. In this configuration, when a spare is taken by a spare gate, it becomes unavailable (i.e. essentially seen as failed) to the rest of the spare gates. The FDEP gate is comprised of a trigger event and a set of dependent components. When the trigger event occurs, it causes the dependent components to become inaccessible or unusable (i.e. essentially failed). The FDEP gate's output is a 'dummy' output (i.e. it is not taken into account during the calculation of the system's failure probability). Along with

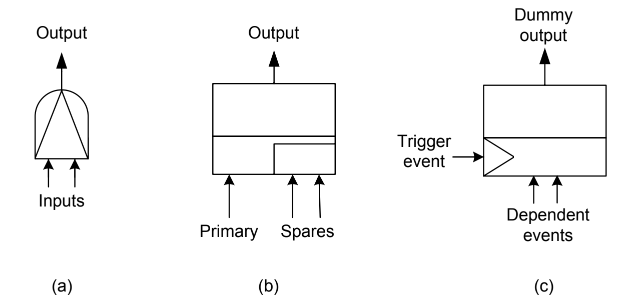

Figure 1. Dynamic gates: (a) PAND, (b) spare, (c) FDEP.

static and dynamic gates, DFT also possess basic events, which are leaves of the tree. A basic event usually represents a physical component having a certain failure probability distribution (e.g. exponential). A DFT element has a number of operational or failed states. In the case of a BE[5], operational states could be further classified as *dormant* or *active* states. A dormant state is a state where the BE failure rate is reduced by a factor called the dormancy factor α. An active state is a state where the BE failure rate λ is unchanged. Depending on the value of α, we classify BE as: cold BE ($\alpha = 0$), hot BE ($\alpha = 1$), and warm BE ($0 < \alpha < 1$). The dormant and active states of a BE correspond to dormant and active modes of the physical component. For instance, a spare tire of a car is initially in a dormant mode and switches to an active mode when it is fixed on the car for use.

[4]A fourth gate called 'Sequence Enforcing' gate has also been defined in [10]; however, it turns out that this gate can be emulated using a cold spare gate.

[5]Also a spare gate as we will see in Section 6.

Galileo DIFTree [11] was the first package to introduce, use, and analyze DFT. DIFTree uses a modular approach to analyze a DFT. Indeed, the DFT is first split into independent static and dynamic modules, the modules are then solved separately and each of them is replaced by a BE with a constant failure probability. The modules' solutions are then combined to find the overall system reliability. This process is iterative as independent modules could be nested. An independent module is dynamic if it contains at least one dynamic gate, otherwise it is static. Static modules are solved using binary decision diagrams and dynamic modules are solved by converting them into Markov chains.

Note that when an independent module is replaced by a BE with a constant failure probability, some information (i.e. the shape) of the module's failure distribution is lost since it is replaced by a single failure probability value. Moreover, since any dynamic gate requires the knowledge of the entire failure probability density functions of their inputs, solving an independent module and replacing it by a BE with a constant failure probability is only possible if the module is part of a larger static (and not dynamic) module. This constraint, which is linked to issue (2) mentioned in the Introduction, makes DFT far less modular (cf. Section 5).

3. Input/output interactive Markov chains

Input/output interactive Markov chains [4] are an integration of input/output automata [16] and CTMC [15, 18]. I/O-IMC are closely related to Interactive Markov Chains [12] (IMC) which are an integration of interactive processes [17] and CTMC.

Figure 2.a shows two examples of input/output interactive Markov chains. Circles denote states in the model and transitions are depicted as arrows. The starting state is identified by a black dot. There are two different kinds of transitions in an I/O-IMC model: *Markovian transitions*, denoted by a small rectangle on the arrow and *interactive transitions*, denoted by a line on the arrow. Each I/O-IMC has an *action signature*, written next to its starting state, which shows how it communicates with the environment. I/O-IMC B, for instance, has an input action $a?$, an output action $b!$ and no internal actions. When each of the I/O-IMC's actions has at least one associated transition, the action signature can be (and often is) omitted. The difference between inputs, outputs and internal actions will be discussed later in this section.

I/O-IMC B has a Markovian transition from state 1 to state 2. This transition has a *rate* of λ. Markovian transitions in I/O-IMC behave exactly the same way as Markovian transitions in CTMC: the I/O-IMC moves from state 1 to state 2 after an exponentially distributed delay. An I/O-IMC with only Markovian transitions can thus be interpreted as a CTMC. I/O-IMC B also has an interactive transition from state 1 to state 3 labeled $a?$. This denotes that the move from 1 to 3 is an *input action* named a. Input actions are denoted with a question mark (i.e. $a?$). If some other I/O-IMC performs an *output action* named a while I/O-IMC B is in state 1 then B will move to state 3 immediately. It is important to note that every state of I/O-IMC B has an outgoing input transition named a. This means that B is always ready to respond to an output-action a, even if this does not result in a state-change (when B is in state 3, 4, or 5). For clarity we will omit these transitions (input-actions from a state to itself) from now on. We say that I/O-IMC B is *input-enabled* with respect to action a. Note that input actions are delayable, i.e. they must wait until another I/O-IMC performs the corresponding output-action.

A different kind of interactive transition from state 4 to state 5 is also present in B. This transition is labeled $b!$ and is an output action. Output actions are denoted with an exclamation mark (i.e. $b!$). When I/O-IMC B performs this output action all I/O-IMC which have b as an input action must perform this input action. Unlike input actions, output actions are immediate; i.e. when I/O-IMC B moves to state 4 no *time* passes before it moves to state 5. It is however possible that another interactive transition is taken immediately. Specifically, if two or more different output actions are possible in a state, then the choice between the transitions is *non-deterministic*. One of the transitions is taken immediately, but it is not known how this choice is made.

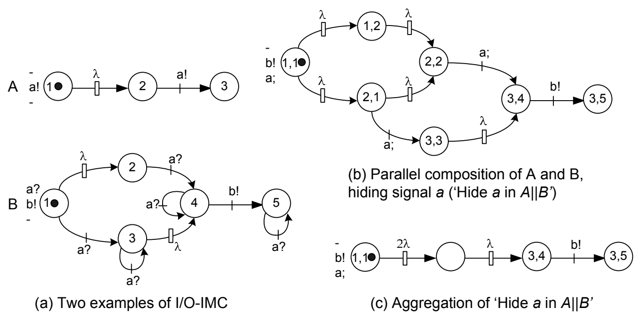
A
a!
B
b!
a?
(a) Two examples of I/O-IMC
(b) Parallel composition of A and B, hiding signal a ('Hide a in A||B')
(c) Aggregation of 'Hide a in A||B'
λ
2λ
a;
b!

Figure 2. Composition, hiding, and aggregation.

Besides input and output actions there are also internal actions. Internal actions are denoted with a semi-colon (;) and model internal computation steps of the system they represent. Thus, internal actions do not influence other I/O-IMC and are not influenced by other I/O-IMC. Similar to output actions, internal actions are immediate.

The reason it is interesting to combine Markovian and interactive transitions is that interactive transitions enable the construction of large I/O-IMC by composition of several smaller I/O-IMC [12]. The subject at hand (the analysis of dynamic fault trees) is a good example. Instead of transforming the entire DFT into one large CTMC we transform the basic events and gates of the DFT first and then create a single I/O-IMC by combining the smaller ones (see Section 5). The I/O-IMC formalism is one such approach to combining Markovian and interactive transitions. A discussion on different approaches to combining Markovian and interactive transitions in one formalism can be found in [12]. An I/O-IMC can also be transformed into a smaller *aggregated* I/O-IMC that is equivalent (i.e. preserving the system reliability measure) to the original I/O-IMC. This state space aggregation, which generalizes the notion of *lumping* in CTMC, can very effectively reduce the resources necessary to create a model of a real-life system [14]. In this work we have used *weak bisimulation* to aggregate the I/O-IMC. For the definition of weak bisimulation for I/O-IMC we refer the reader to [4] and for details on the complexity of the minimization algorithm we refer to [12]. Figure 2 shows an example of how two I/O-IMC A and B can be composed (and hiding signal a with which they communicate) and how the resulting I/O-IMC can then be aggregated. When composing I/O-IMC A and B we *synchronize* on signal a, because it is in both their action signatures. Since B has a as an input, it has to wait for A's output action $a!$. This explains the absence of an input transition $a?$ from state $(1,1)$ in the composed model. However, in state $(2,1)$, for instance, A outputs its signal a (and moves to state 3) and B simultaneously makes the corresponding input transition and moves from state 1 to 3. All Markovian transitions and non-synchronizing signals are essentially interleaved during composition. Since weak bisimulation abstracts from internal (unobserved) actions; states (1,2), (2,1), (2,2), and (3,3) are equivalent given that they essentially all move with a rate λ to the same state (3,4). Indeed, these 4 states are aggregated into a single (unlabeled) state in Figure 2.c.

4. DFT to I/O-IMC conversion

During the conversion of a DFT to a MC, the DIFTree algorithm [11] proceeds as follow: First, the MC's initial state is created, listing the states of all basic events contained in the DFT as operational [6]. From the initial state, every BE is being failed (according to its failure rate) one at a time and the corresponding transition and next state are created where the state information (i.e. operational or failed) of the basic event is updated. For every newly created state, the DFT model (i.e. system state) is evaluated to determine whether the state corresponds to an operational or a failed system state[7]. As long as a state is an operational state, every operational BE contained in that state is being failed, and a corresponding new transition (and optionally a new state) is created. Note that each MC state has a vector list-

[6]Some extra information, such as which spare gate is using a given spare, is also appended to the state.

[7]This operation is unnecessary in the I/O-IMC framework.

ing the state of all basic events contained in the DFT; consequently, this makes the state-space grow exponentially with the number of basic events.

This DIFTree MC generation approach, where the model of a dynamic system is generated at once and as a whole, is to be contrasted with our compositional aggregation approach. paper In our I/O-IMC framework, each DFT element (i.e. basic event and gates) has a corresponding I/O-IMC precisely defining its behavior (i.e. semantics). Every I/O-IMC has an initial operational state (i.e. with no incoming transition), some intermediate operational (dormant or active) states, a *firing* (i.e. about to output a failure signal) state, and an absorbing *fired* state. The firing and fired states are both failed states and are drawn as gray circles and double circles respectively. There are two main signals (or actions): a *firing signal* and an *activation signal*. The firing signal of element A is denoted by f_A and it signals the failure of a BE or a gate. The activation signal refers to the activation (i.e. switching from dormant to active mode) of a spare A and is denoted by a_A. An activation signal is only output by spare gates, and $a_{A,B}$ denotes the activation of spare A by spare gate B. Indeed, since a spare A can be shared, and thus activated, by multiple spare gates, an activation signal is needed for each of the spare gates. These activation signals are then translated by an auxiliary I/O-IMC model[8] called activation auxiliary (AA) into a single activation signal a_A which acts as an input to the spare A. In the original DIFTree methodology, only BE can act as spares, and thus BE are the only elements that exhibit a dormant as well as an active behavior. However, in our framework we lift this restriction by allowing any independent sub-system to act as a spare. As a consequence, spare gates also exhibit dormant and active behaviors (see Section 6 for further details).

In the following, we show the I/O-IMC of the basic event, the PAND gate, the FDEP gate, and the spare gate (the full details on all the gates can be found in [4]). We postpone the discussion on the spare gate model until Section 6.

4.1. Basic event I/O-IMC model

As pointed out in Section 2, a basic event has a different failing behavior depending on its dormancy factor. For this reason we identify three types of basic events and correspondingly three types of I/O-IMC. Figure 3 shows the I/O-IMC corresponding to a cold, warm, and hot basic events (all called A). The I/O-IMC clearly captures the behavior of the basic event described in Section 2.

[8]The AA model is essentially an OR gate having as inputs the various activations signals coming from the spare gates, and as an output a spare activation signal rather than a firing signal.

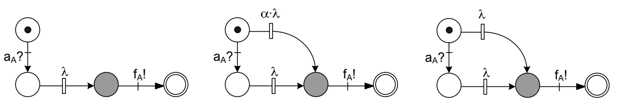

Figure 3. I/O-IMC models of cold, warm, and hot BE.

4.2. PAND gate I/O-IMC model

The PAND gate fires if all its inputs fail and fail from left to right order. If the inputs fire in the wrong order, the PAND gate moves to an operational absorbing state (denoted with an X on Figure 4). Figure 4 shows the I/O-IMC

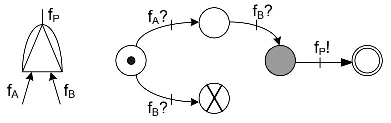

Figure 4. I/O-IMC of the PAND gate.

of the PAND gate P with two inputs A and B (A being the leftmost input).

4.3. FDEP gate I/O-IMC model

A functional dependency is modeled using a firing auxiliary (FA). The FA governs when a dependent DFT element fires, i.e. either when the element fails by itself or when its failure is triggered by the FDEP gate trigger. There exists a different FA for each dependent event. Figure 5 shows the FA of element A, which is functionally dependent upon B. The signal f_A^* corresponds to the failure of element A by itself without factoring in its functional dependency (i.e. in isolation), and the signal f_A corresponds to the failure of A when also considering its functional dependency upon B. In order to get the correct behavior of the element A, one has to compose the three I/O-IMC corresponding to A in isolation, to its FA, and to the trigger B. Note that any element which has A as input has to now interface with A's FA rather than directly with A. Note also that the firing auxil-

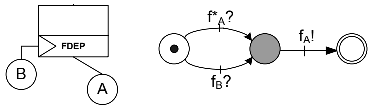

Figure 5. I/O-IMC of the firing auxiliary.

iary I/O-IMC is similar to the OR gate I/O-IMC with two input signals f_A^* and f_A.

In the original DIFTree methodology, only BE can be dependent events. However, in our framework we lift this restriction by allowing any sub-system to be an FDEP gate dependent event (Section 6).

The I/O-IMC models discussed above have been generalized (cf. [9] for details) to deal with any number of inputs.

4.4. Simultaneity and non-determinism

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 Figure 6, 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 configuration a) and which spare gate gets the shared spare S (in configuration b)? These examples were also discussed in [8], and we believe that this is an inherent non-determinism in these models. In [8], these special cases are dealt with by systematically removing the non-determinism by transforming it into a probabilistic (or deterministic) choice. In our framework, we allow non-determinism and naturally provide a mechanism for detecting it should this arise in a particular DFT configuration. Moreover, if the non-determinism was not intended, then its detection indicates that an error occurred during the model specification. Non-determinism could also be an inherent characteristic of the system being analyzed, and should therefore be explicitly modeled. An example of such a system would be a repairman following a first failed first repaired policy and being in charge of two components. Now, if both components fail at the same time, then we might decide to model the choice of which one to pick first for repair to be a non-deterministic choice made by the repairman.

In the I/O-IMC formalism, the DFT configurations depicted in Figure 6 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

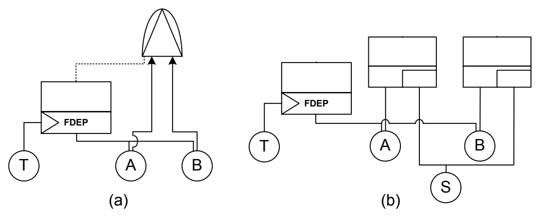

Figure 6. The occurrence of non-determinism.

fail in a non-deterministic order (i.e. essentially consider all combinations of ordering). In this case, the final I/O-IMC model is not a continuous-time Markov chain (CTMC) but rather a continuous-time Markov decision process (CTMDP), which can be analysed by computing bounds of the performance measure of interest (see [2] for an efficient algorithm on analysing CTMDP).

4.5. Conversion of a DFT into a community of I/O-IMC

We have defined the individual I/O-IMC models for each of the DFT elements and some were described in the previous sub-sections. We can now convert any given DFT into a corresponding set of I/O-IMC models. Moreover, we need to match the inputs and outputs of all the models. The mapping between the DFT and the I/O-IMC community is a *one-to-one mapping*, except for some cases (e.g. spare activation and functional dependency) where extra auxiliary I/O-IMC are also used.

5. DFT analysis

Once the DFT has been converted into an I/O-IMC community, the compositional aggregation methodology can be applied on the I/O-IMC community to reduce the community to a single I/O-IMC. The final I/O-IMC reduces in many cases to a CTMC[9]. This CTMC can be then solved using standard methods [18] to compute performance measures such as system unreliability. The full conversion/analysis algorithm[10] 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 an I/O-IMC community.
2. Pick two I/O-IMC and parallel compose them.
3. Hide output signals that won't be subsequently used (i.e. synchronized on).
4. Aggregate (using weak bisimulation as mentioned in Section 3) the I/O-IMC obtained from the composition of the two I/O-IMC picked in Step 2 and the hiding of the output signals in Step 3.
5. Go to Step 2 if more than 1 I/O-IMC is left, otherwise go to Step 6.
6. Analyse the aggregated CTMC (or CTMDP).

5.1. Example: The cardiac assist system

The cardiac assist system (CAS) model is taken from [3] and is based on a real system. The DFT is shown in Figure 7. The CAS consists of three separate and distinct modules: The CPU unit, the motor unit and the pump unit.

[9]Occasionally to a CTMDP if some non-determinism remains.
[10]Note that this algorithm is amenable to parallelization.

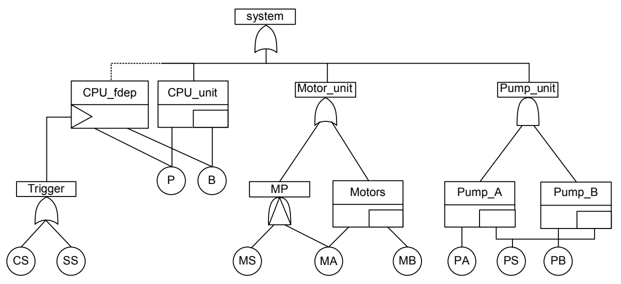

Figure 7. The cardiac assist system DFT.

There are two CPUs: a primary (P, $\lambda = 0.5$) and a warm spare (B, $\lambda = 0.5$) with $\alpha = 0.5$. Both are functionally dependent on a cross switch (CS, $\lambda = 0.2$) and a system supervision (SS, $\lambda = 0.2$), which means that the failure of either these components will trigger the failure of both CPUs. There are also two motors: a primary (MA, $\lambda = 1$) and a cold spare (MB, $\lambda = 1$). The switching component (MS, $\lambda = 0.01$) turns on the spare motor when the primary fails. The MS is also subject to failure, but this failure is only relevant if it occurs before the failure of the primary motor. Finally, there are three pumps: two primary pumps (PA and PB with $\lambda = 1$ for both) running in parallel and a cold shared spare pump (PS, $\lambda = 1$). All three pumps must fail for the pump unit to fail.

We have developed our own conversion tool which takes as input a DFT specified in the Galileo DFT format [11], and translates the DFT into its corresponding community of I/O-IMC models in the format of the TIPP tool [13]. The I/O-IMC models are then composed and aggregated using the TIPP tool. Finally, the system unreliability is computed also using the TIPP tool. Each of the aggregated I/O-IMC models of the three modules had 6 states. This result was comparable to the Galileo tool results, where the biggest generated CTMC (the pump unit) had 8 states. The system unreliability obtained using the TIPP tool was 0.6579 for a mission time equals to 1 time unit. The result provided by the Galileo DIFTree tool was identical. In the next section, we show, through a second example, the enhanced modular analysis that we attain using the I/O-IMC framework.

5.2. Modular analysis

In this section, we illustrate the lack of modularity (already pointed out in [1, 5] and which leads to a worsening of the state-space explosion problem) in the DIFTree methodology with respect to dynamic modules. The example at hand, shown in Figure 8, is called the cascaded PAND system (CPS) for which a variation can be found in [5]. The CPS consists of two PAND gates and three AND gates each having four identical BE with a failure rate equals to 1. In fact, the three AND gates constitute independent and identical modules. However, since the top gate is a dynamic gate, the DIFTree methodology does not modularize the tree into five[11] distinct modules; but it rather considers the whole tree as a single module. The reason that DIFTree does not consider, for instance, module A as an independent module is because its parent gate (i.e. the PAND gate *System*) is a dynamic gate (cf. Section 2).

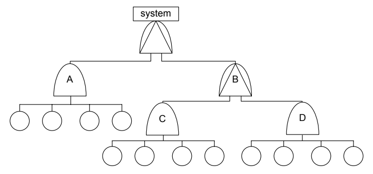

Figure 8. The cascaded PAND system.

Thanks to the interactivity of I/O-IMC, we are able to further modularize the CPS and generate the corresponding I/O-IMC for each of the five modules. Moreover, since A, C, and D are identical, we only need to generate the I/O-IMC for one of these modules and reuse it by renaming some of the activation and firing signals. Figure 9 shows the I/O-IMC of module A after parallel composition and aggregation. The I/O-IMC is particularly small because all basic events have the same failure rate and the order in which they fail is irrelevant. Solving the CPS following this modular compositional aggregation analysis technique resulted in 156 states and 490 transitions for the biggest generated I/O-IMC. This result is to be contrasted with the DIFTree solution which resulted in 4113 states and 24608 transitions. The system unreliability, for a mission time equals to 1 time unit, is the same in both cases and equals 0.00135. The reason DIFTree performs so poorly is because the corresponding CTMC is generated for the whole tree (i.e. with 12 basic events) and at once, and in which even irrelevant failure orders (such as for the BE belonging to module A) are accounted for. The compositional aggregation approach performs particularly well for this example due to the high modularity of the system. However, the approach does not perform as well for some examples we have worked on where the DFT elements are highly connected (i.e. numerous interdependencies/interactions between DFT elements), which leads to the incapacity to effectively divide the system into independent small modules.

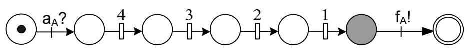

Figure 9. I/O-IMC of module A.

[11]Each gate acts as an independent module.

6. Modular model-building

Static fault trees are highly modular, i.e., any sub-tree can be used as an input to another static gate. Unfortunately, this modularity does not currently apply to dynamic trees. Indeed, only BE are allowed as inputs to spare gates and as dependent events of FDEP gates. In the I/O-IMC framework, we increase the modularity of DFT by allowing: (1) independent sub-trees to act as primary and spare components and (2) FDEP gates to trigger any arbitrary element (BE and gates).

This section and the CPS example of the previous section show the enhanced modularity obtained in our framework and the ability to reuse, without restrictions, independent sub-modules within larger dynamic modules. Such reusability, which was previously only fully implementable in static FT, is a very powerful and useful concept in large FT. Indeed, being able to 'plug-in' modules is a practical feature when designing very large systems where the model is build incrementally and/or various teams are working on different parts of the system.

6.1. Spare modules extension

The system depicted in Figure 10.a is a typical system we would like to be able to model using the DFT formalism. The primary and spare components are not BE, but rather more complex sub-systems. In the I/O-IMC framework, we allow primary and spare components to be any independent sub-system[12]. We enforce the independence restriction because otherwise the activation of these components becomes unclear.

This extension of primary and spare components requires the reexamination of the concept of activation. The intuition is as follows: In Figure 10.a, the activation of module 'spare' simply means the activation of the two BE C and D. The module's (represented by its top-node AND gate) dormancy is defined by the dormancy of its BE. The AND gate I/O-IMC model is not changed and has the same behavior whether 'spare' is dormant or active. In fact, whenever an activation signal is received by module 'spare', this same activation signal is simply passed on to the next components (which happen to be BE in this example), one level down the tree. The behavior of all the gates (i.e. I/O-IMC models) is unchanged whether they are used as spares or not. However, the spare gate is an exception to this rule and does behave differently when used as a spare. Figure 10.b illustrates this: When 'spare' is not activated (i.e. 'primary' has not failed), BE C and D are dormant; and even if C (being a warm

[12]A sub-system is usually named after its top-node and is independent if (1) all the elements in the tree have inputs from only elements within the same tree and (2) all the outputs, except for the top-node, are also within the tree and therefore hidden to the rest of the system

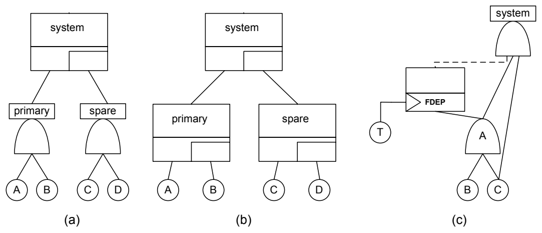

Figure 10. Complex spares and FDEP gate extension.

spare) fails, D remains dormant. This is the same behavior as with the 'spare' AND gate in Figure 10.a. If 'spare' is activated, the activation signal is only passed to the primary C and D remains dormant (this is clearly different from the AND gate where both BE are activated). Should C fail and 'spare' being in its active state, then D is activated. Based on the above explanation, Figure 11 shows the behavior of the spare gate A[13]. Signals $a_{S,A}$ and $a_{S,C}$ are actions out-

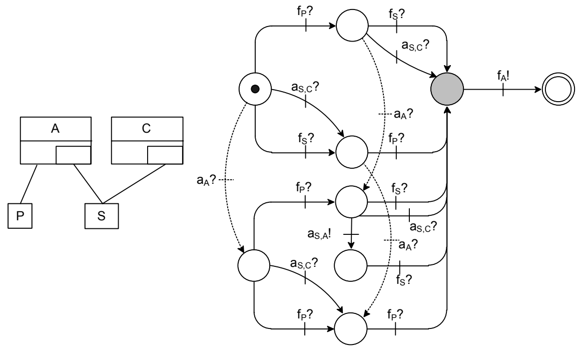

Figure 11. The spare gate I/O-IMC model.

put respectively by A and C signaling that the spare S has been taken[14]. The spare gate I/O-IMC model has been, of course, generalized to handle multiple spare gates sharing multiple spares (i.e. the most general case).

6.2. FDEP gate extension

In this framework, the FDEP gate can trigger the failure of any gate (representing a sub-system) and not only BE. 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 Figure 5. Figure 10.c shows such a configuration where T triggers the failure of the sub-tree A. Note that sub-system 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.

[13]For clarity, the activation signal is drawn as a dashed line.
[14]This solution is not very scalable since it suggests that all spare gates sharing a spare communicate with each other. A better solution has been found where a 'spare granting' auxiliary is used.

7. DFT elements extension

In this section, we show, through some examples, how readily one can extend the DFT elements within the I/O-IMC framework. In fact, 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 is kept as local as possible. These extensions only affect Step 1 of the DFT conversion/analysis algorithm laid out in Section 5. The remaining five steps, including the composition, the aggregation and the analysis remain unchanged. The first extension concerns the modeling of *inhibition* and *mutually exclusive events*. The second extension is somewhat more involved and concerns the modeling of repair.

7.1. Inhibition and mutual exclusivity

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 4.3), this could be modeled by simply adding an *inhibition auxiliary* (IA). Figure 12 shows the configuration of such inhibition and the corresponding I/O-IMC model of the IA of B. Signal 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.

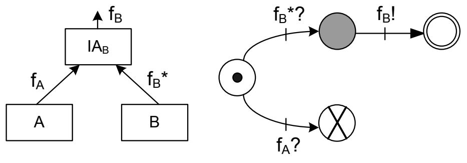

Figure 12. The I/O-IMC model of the IA.

If event B also inhibits the failure of A, then we need to add an IA for A as well. In this way, the failure of A and the failure of B become two mutually exclusive events. Mutual exclusivity is very useful when modeling a component exhibiting various failure modes. A typical example is a switch with two failure modes: 'failing to close' and 'failing to open'. These failures have normally different probabilities of occurrence and different consequences on the overall system. The switch failure modes have to be modeled as two mutually exclusive BE since the switch can either fail open or fail closed, but not both.

7.2. 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. Due to the lack of space, 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 on Figure 13. Here, μ denotes the BE repair rate and $r!$ is a signal output by the BE notifying, to the rest of the elements, that it has been repaired. Note that the fired state is not absorbing anymore. As an alternative model, one can of course think of the BE interacting with a repair station (RS); in which case, the repair process[15] would be part of the RS I/O-IMC model and f would also be an input to the RS. An extra signal (input to the BE and an output of the RS) would also be needed for communication between the BE and RS and signaling that the RS has finished the repair. The repairable AND

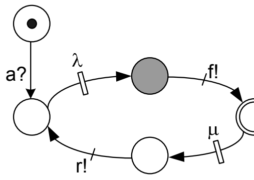

Figure 13. The repairable BE I/O-IMC model.

gate I/O-IMC model is shown on Figure 14. 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, Figure 14 has 3 extra states. If we

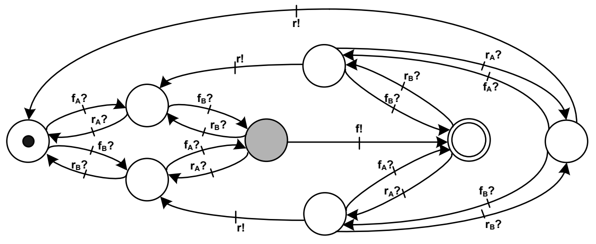

Figure 14. The repairable AND gate I/O-IMC model.

consider a very simple repairable system composed of an AND gate with two BE A and B (Figure 15.a), then the resulting I/O-IMC after automatic composition and aggregation[16] is, as expected, a CTMC shown on Figure 15.b.

[15]Which could be more complicated than a single Markovian transition with repair rate μ.

[16]And abstraction of the AND gate's activation and failure signals.

At this point, one can perform some analysis on the CTMC

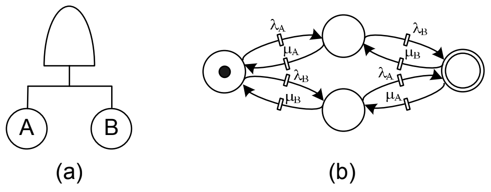

Figure 15. A simple repairable system.

such as computing the system unavailability.

8. Conclusion and future work

In this paper, we have illustrated the use of the I/O-IMC framework for the analysis of DFT and showed, through some examples, the increase of 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.

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) Generalize the concept of activation to any type of mode switch[17]; this is similar to the notion of 'triggered Markov processes' defined in [6]. (3) In this paper, we have only considered exponential failure distributions for BE; it would be worthwhile investigating the use of phase-type distributions, which naturally integrate into the I/O-IMC framework, to approximate any BE failure probability distribution.

References

[1] S. Amari, G. Dill, and E. Howald. A new approach to solve dynamic fault trees. In *Annual Reliability and Maintainability Symposium*, pages 374–379, January 2003.

[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. *Theor. Comput. Sci.*, 345(1):2–26, 2005.

[3] H. Boudali. *A Bayesian network reliability modeling and analysis framework*. Phd dissertation, University of Virginia, Charlottesville, VA, May 2005.

[4] H. Boudali, P. Crouzen, and M. I. A. Stoelinga. A compositional semantics for Dynamic Fault Trees in terms of Interactive Markov Chains. Technical report, University of Twente, Enschede, the Netherlands, to appear.

[5] H. Boudali and J. B. Dugan. A new Bayesian network approach to solve dynamic fault trees. In *Reliability and Maintainability Symposium*, Jan 2005.

[17]In this respect, inhibition can be viewed as a mode switch where the inhibited event moves to a permanent operational state upon the receipt of the failure signal from the inhibitor.

[6] M. Bouissou and J.-L. Bon. A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes. *Reliability Engineering and System Safety*, 82(2):149–163, 2003.

[7] M. A. Boyd. *Dynamic fault tree models: techniques for analyses of advanced fault tolerant computer systems*. Phd dissertation, Dept. of Computer Science, Duke University, 1991.

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

[9] P. Crouzen. Compositional analysis of dynamic fault trees. MSc thesis, University of Twente, Enschede, the Netherlands, 2006.

[10] J. B. Dugan, S. J. Bavuso, and M. A. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–377, September 1992.

[11] J. B. Dugan, B. Venkataraman, and R. Gulati. DIFTree: a software package for the analysis of dynamic fault tree models. In *Reliability and Maintainability Symposium*, pages 64–70, Jan 1997.

[12] H. Hermanns. *Interactive Markov Chains*, volume 2428 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[13] H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the TIPPtool. *Lecture Notes in Computer Science*, 1469:51–62, 1998.

[14] H. Hermanns and J.-P. Katoen. Automated compositional Markov chain generation for a plain-old telephone system. *Science of Computer Programming*, 36(1):97–127, 2000.

[15] R. A. Howard. *Dynamic probability systems. Volume 1: Markov models*. Decision and Control. John Wiley & Sons, Inc., 1971.

[16] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1988.

[17] R. Milner. *Communication and Concurrency*. Prentice Hall Inc., 1989.

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

[19] K. K. Vemuri, J. B. Dugan, and K. J. Sullivan. Automatic synthesis of fault trees for computer-based systems. *IEEE Transactions on Reliability*, 48(4):394–402, December 1999.

[20] W. E. Veseley, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook, NUREG-0492*. United States Nuclear Regulatory Commission, NASA, 1981.