Compositional synthesis of Temporal Fault Trees from State Machines

Nidhal Mahmud
Dept. of Computer Science
University of Hull, Hull, UK
N.Mahmud@hull.ac.uk

Martin Walker
Dept. of Computer Science
University of Hull, Hull, UK
Martin.Walker@hull.ac.uk

Yiannis Papadopoulos
Dept. of Computer Science
University of Hull, Hull, UK
Y.I.Papadopoulos@hull.ac.uk

ABSTRACT

Dependability analysis of a dynamic system which is embedded with several complex interrelated components raises two main problems. First, it is difficult to represent in a single coherent and complete picture how the system and its constituent parts behave in conditions of failure. Second, the analysis can be unmanageable due to a considerable number of failure events, which increases with the number of components involved. To remedy this problem, in this paper we outline an analysis approach that converts failure behavioural models (state machines) to temporal fault trees (TFTs), which can then be analysed using Pandora — a recent technique for introducing temporal logic to fault trees. The approach is compositional and potentially more scalable, as it relies on the synthesis of large system TFTs from smaller component TFTs. We show, by using a Generic Triple Redundant (GTR) system, how the approach enables a more accurate and full analysis of an increasingly complex system.

Keywords

Dependability Analysis, State Machines, Temporal Fault Tree synthesis, Pandora

1. INTRODUCTION AND BACKGROUND

Dependability analysis of dynamic systems, where the order in which events occur can affect the overall outcome (e.g. total failure), remains highly challenging. One central issue in modern safety-critical systems is that they are becoming increasingly dynamic and, correspondingly, more complex to analyse. Thus it becomes ever more important to find scalable approaches that allow for rapid analysis while still being able to take into account the dynamics of such systems.

Finite state transition models (e.g. SMs) have become a prevalent paradigm for the description of dynamic systems, thanks to their flexibility and ease of use; such models help designers and analysts to better understand how these systems can fail. SMs are used in several approaches, e.g., an extension to Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) described in [1], Altarica [2], FSAP/NuSMV-SA [3] and, most recently, AADL [4].

There have been several attempts to assess systems modelled with SMs for the purposes of dependability analysis. One approach consists of converting SM models to Generalised Stochastic Petri Nets (GSPNs), which are then analysed for dependability. This is one of the approaches that can be applied to AADL system models, as described in [5] — AADL error models are effectively state automata showing transitions from normal to degraded and failed states [4]. Yet in this approach, it can be problematic to perform qualitative analysis, i.e., establishment of full causal relationships between causes and effects of failure, as in traditional analysis methods like Failure Mode and Effects Analysis (FMEA). Qualitative analysis is particularly important when probabilistic data are not available, e.g. at early stages of design; it is useful to be able to identify and solve potential design problems early, rather than in later iterations when the design is more concrete and changes are more expensive. Decisions made at these early stages can be critical for determining the future shape of the system, and so it is important that safety can be taken into account at all stages of the design process.

An alternative approach involves conversion of SMs to fault trees. Fault trees are logical networks of events that show how combinations of failures can cause a given system failure and are ideally suited for qualitative analysis. However, there are difficulties with this type of conversion; in particular, the temporal semantics of SMs (which are dynamic models) are lost in the translation to combinatorial fault trees (which are static models), and this can potentially cause serious errors/inaccuracies in the analysis (e.g. when the sequencing of faults affects the outcome). There have been some efforts made to work around this issue; for example, in [2], NOT gates were incorporated into the conversion to fault trees to indicate that some events did not occur. Although this prevents a conjunction of two mutually exclusive SM paths occurring as an analysis result, it still cannot distinguish between paths that differ only in sequence - e.g. two faults which lead the system into two mutually exclusive states depending on which fault occurred first.

The conversion of SMs to fault trees has been applied to both AADL models [6] and to Altarica descriptions of systems [2], but in neither approach are the temporal semantics retained during transformation. AADL is gaining growing acceptance in the aerospace community and Altarica's modelling and analysis platform — a tool which uses computer-performed generation of fault trees — was qualified as a validation tool in several aerospace projects, including Airbus civil aircraft programs [7], so solving this problem of losing the temporal semantics of SMs could potentially have a positive impact in such areas.

To remedy the problem of converting dynamic models to static fault trees, an approach which consists of generating

temporal fault trees (TFTs — which are dynamic, rather than static) from SMs has been described in [8]. The technique mainly uses a Priority-OR (POR) gate to differentiate sequences of faults. The gate can represent situations where one event takes priority over others and must occur first, but without specifying that the other events must also occur - e.g. A POR B is true if A occurs and B does not, or if A occurs and B occurs but A occurs first.

In this paper, we extend this technique to enable a full compositional synthesis of TFTs for Pandora analysis. The fault trees are generated from the SMs of the components of a system, and the approach works backwards, starting with the TFTs of the system failures (i.e., total failures as top events). For each TFT event that is an input deviation (i.e., a fault or failure propagated to the input of a component, typically represented by the top event of another fault tree in another component), the synthesis method joins the TFTs together to form a single, full fault tree for each system failure. These can then be analysed to determine the root causes of the failures without sacrificing any information about the dynamic behaviour of the system. In principle, the method can also be combined with a compositional analysis technique like HiP-HOPS, which would enable automatic synthesis and analysis of TFTs from complex models of systems in which the error behaviour is described with state automata.

Section 2 of this paper provides more information about temporal fault tree analysis and Pandora, while section 3 is devoted to describing the algorithm for the automatic generation of TFTs from SM models of systems. In section 4, we present our approach for compositional synthesis of TFTs. We demonstrate the synthesis method by using a generic triple redundant (GTR) example system and show how the resultant TFTs can be analysed with Pandora. Finally, in section 5 we present our conclusions and thoughts on future work.

2. TEMPORAL FAULT TREE ANALYSIS

2.1 Context

A fault tree is a combinatorial model based on Boolean logic, showing the relationships between combinations of basic events (typically component failures) and the system-level failures they cause. Events are combined via logical gates, such as AND gates (represented in this paper by '.') and OR gates (represented by '+'). Synthesis of a fault tree is a deductive process, beginning with the top event — the system failure being analysed — and investigating its intermediate causes at each step until the root causes — the basic events — are reached. Once constructed, a fault tree can be qualitatively analysed by manipulating its Boolean structure to obtain the minimal cut sets — the smallest sets of necessary and sufficient events that cause the top event.

Fault trees are a very effective analysis technique in many situations, but they are ill-suited for modelling dynamic systems. In particular, they cannot model the effects of two (or more) different sequences of the same events, a situation frequently exhibited by dynamic systems.

The requirement for an extension to fault tree analysis to enable the modelling and analysis of dynamic systems is not a new one. The Priority-AND (PAND) gate — probably the earliest attempt at solving this problem — dates back to at least 1976 [9]. The PAND gate is true only if all of its events occur and if they occur in a specific order (usually left to right).

More modern approaches tend to fall into one of two categories: one set of solutions focuses on increasing the number and expressiveness of the fault tree gates by adding new 'temporal' operators, while the other set instead focus on adding additional information or constraints to the events of the fault trees. The most prominent example of the former category is the Dynamic Fault Tree (DFT) approach [10]. DFTs are intended to analyse dynamic systems featuring functional dependencies, sequences of events, and spare components. DFTs are primarily intended for quantitative analysis, and this is achieved by means of a conversion of the fault tree — or at least those parts of it featuring dynamic gates [11] — into a Markov model, which can then be evaluated. While effective in many cases, Markov models can be subject to combinatorial explosion, and so other methods have been proposed for the quantification of DFTs, including numerical integration [12] and Monte Carlo techniques [13]. However, there has been less focus on qualitative analysis of DFTs, which often restricts their usage to situations where probabilistic data is available, although progress is being made in this area, e.g. the algebraic approach developed in [14], which allows both quantitative and qualitative analysis of DFTs.

As an example of the second category of approaches, Gorski & Wardzinski [15, 16] introduce the idea of an enabling condition to fault trees, which is an additional intermediate event that describes a temporal constraint. A formal specification known as the Common Safety Description Model, or CSDM, is used to formulate the enabling conditions. In this approach, events are not treated as instantaneous occurrences (as they are in traditional fault tree semantics); instead, they are defined to be "distinguished states of a system which can last for some time." Whereas DFTs sought to enable quantitative analysis of dynamic systems, the CSDM (and similar approaches) is intended primarily to allow fault trees — with more rigorous semantics and definitions — to be used as part of a formal specification of a system. Although qualitative and quantitative analysis is possible, the fault tree is analysed using classical methods, and it is only once the minimal cut sets have been obtained that the temporal constraints are considered. These can then be used in addition to the cut sets to produce a Timed Petri Net to determine reachability of the top event given the temporal constraints described by the enabling conditions. Therefore the fault tree analysis is not the goal in itself but merely a preliminary step in the overall process.

2.2 Pandora

One of the more recent attempts to extend fault trees with dynamic or temporal information for the purposes of qualitative analysis is Pandora [17]. Pandora is a gate-based approach and introduces three new "temporal gates" to overcome the limitations of fault trees regarding the sequencing of events: the Priority-AND (PAND) gate (symbol '<'), which represents a sequence of events; the Simultaneous-AND (SAND, '&') gate, which represents simultaneous occurrence of events, e.g. as a result of a common trigger; and the Priority-OR (POR, symbol '|'), which as described earlier models a priority situation where one event must occur first and other events may or may not occur subsequently.

Pandora is designed for temporal qualitative analysis in

the same way that static fault trees can be analysed qualitatively through the use of Boolean logic. Therefore, just as Boolean logic has logical laws, Pandora has "temporal laws". Some of Pandora's temporal laws are derived from Boolean laws, whilst others are new. Temporal laws can be proved by means of Temporal Truth Tables (TTTs), which are temporal analogues of Boolean truth tables and which show equivalence between two or more logical expressions.

Temporal expressions are generally complex, particularly when compared with Boolean ones; hence the temporal laws are very useful in enabling the minimisation of the expressions and thereby reducing the complexity involved. They also make it possible to perform qualitative analysis by identifying and removing redundant sequences of events.

Pandora allows more than one form of reduction. As well as removal of redundancies, e.g. A<B + A|B ⇔ A|B (where A and B are fault tree events), Pandora also allows reduction via the recognition and removal of contradictions and by means of 'completion' — conversion of temporal expressions into static Boolean expressions. Contradictions are identified by specific laws, e.g. the Laws of Mutual Exclusion indicate that only one temporal ordering can apply to a pair of events at any time, thus A<B . B<A results in a contradiction. Similarly, an expression such as A<A.B + C also contains a contradiction (since A cannot occur before itself); this reduces to 0.B + C and ultimately to just C.

The concept of 'completion' expresses equivalence between temporal and non-temporal expressions and is accomplished by means of the Completion Laws — a set of temporal laws in Pandora that relate non-temporal gates to temporal ones. These laws can be used to reduce complex temporal expressions into much simpler Boolean expressions where possible. The three Completion Laws are:

- Conjunctive: A<B + A&B + B<A ⇔ A.B
- Disjunctive: A|B + A&B + B|A ⇔ A+B
- Reductive: A<B + A&B + B|A ⇔ B or A.B + B|A ⇔ B

For full details about Pandora's temporal laws we refer the reader to [18], but four laws in particular will be useful in this paper:

- (A|B).B ⇔ A<B
- A . A|B ⇔ A|B
- (A|B) . (A|C) ⇔ A|B|C
- A|(B+C) ⇔ A|B|C

These laws all express equivalence, e.g. in the first law, between the POR gate with an additional conjunction (A must occur first, but B must also occur) and the PAND gate (A occurs before B, both events have to occur). Thus they can be used to remove redundant events or parentheses from an expression in order to simplify it.

Different algorithms can be used to minimise temporal expressions, ranging from logical manipulation to hierarchical checking [18], but they all allow Pandora to be used to determine the "minimal cut sequences" (MCSQs) of the fault tree: analogous to Minimal Cut Sets (MCSs), these represent the smallest sequences of events necessary to cause the top event of the fault tree to occur. These results therefore retain the temporal information represented in the fault tree by the PAND, POR, and SAND gates and thus are more accurate and informative than the results of a comparable static FTA.

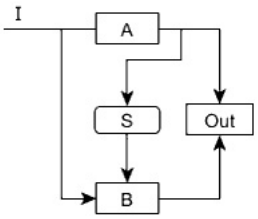

Figure 1: Primary Standby system

3. COMPILATION OF SMs INTO PANDORA EXPRESSIONS

3.1 Motivation

The use of state machines is an expressive, high-level method of modelling the behaviour of systems (or components). SMs readily express the different effects of events on a system in different states, making them well-suited to modelling the effects of failures and fault events on a system.

As mentioned previously, SMs can be converted into fault trees. To transform a SM to a purely combinatorial fault tree, each of the SM final failure states becomes the head of its own fault tree. All paths through the SM to the given final state become a disjunction of possible causes of the top event, and each cause is represented as a conjunction of all failure events in that path which have caused the system, or the component, to enter the final state. Each conjunction represents one cut set of the fault tree, and if it contains no redundancies, then it is a minimal cut set (MCS).

For example, consider Figure 1 (used in [8]) which represents a simple Primary Standby (PS) system with three components: 'A' represents the primary component, while sensor 'S' monitors the output from 'A' and is meant to activate the backup 'B' if it detects an omission of output. Supply must be received from either A or B for proper functioning of the whole system.

For the SM of S in Figure 2, the initial state is 'Monitoring', which means that the sensor is ready and standing by to activate B. If a fault occurs in the output from A (e.g. an output omission represented as 'O-A' in the SM) and is detected, then the state of S alters to 'Omission Detected' (i.e. it needs to activate B). However, the system can continue to function if S fails after waking B up (because B takes over A's function), thus that sequence of failures — O-A occurs first then S fails second — will lead the sensor to a 'Safely Failed' state. By contrast, if S fails first (entering the 'Prematurely Failed' state), then it will be unable to detect a subsequent omission of output from A, and thus this second sequence will lead to a 'Severely Failed' state (since B will not be activated, which means a total failure of the system). This latter state represents an output omission from S ('O-S') i.e., an error propagation through the system (which leads to a complete failure in this particular case).

Therefore, as can be seen from the situation in Figure 2, we have two different sequences of the same two failures each leading to different failure states of the sensor. One sequence can cause system failure, whereas the other will give it a chance to continue to function with the backup. However, transforming this SM to a purely combinatorial fault tree would give a conjunction of the same failures as a

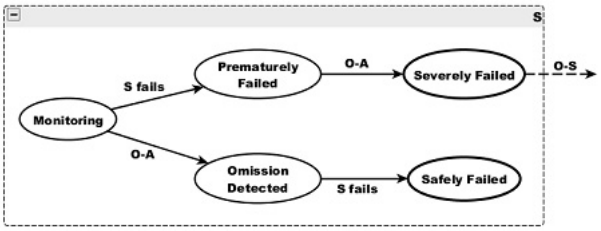

Figure 2: State machine of S

common cause for both 'Safely Failed' and 'Severely Failed' states: Safely Failed = O-A.(S fails) and Severely Failed = (S fails).O-A. Since O-A.(S fails) $\Leftrightarrow$ (S fails).O-A according to the commutative law of Boolean logic, these two conjunctions are logically equivalent. Hence subsequent fault tree analysis would suggest that the same two failures can lead to two different sensor failures — at best a non-deterministic result and at worst a logical impossibility — and any consequent quantitative analysis would also be incorrect. Thus, this kind of transformation can lead to serious errors in the analysis: the results are not an accurate representation of the behaviour shown in the SM. We therefore need some way of representing event sequences within fault trees, and then it would be possible to convert SMs into fault trees without sacrificing the relevant event sequencing, which affects the outcome.

3.2 Representing SMs using Pandora logic

Consider again the two sequences in the example state machine in Figure 2. With Pandora, we can represent these sequences explicitly using the PAND gate: Safely Failed = O-A<(S fails) and Severely Failed = (S fails)<O-A. The PAND gate is not commutative, i.e., O-A<(S fails) $\not\Leftrightarrow$ (S fails)<O-A, and thus the importance of the ordering of the events is preserved, each one leading to a different system failure.

However, not all state machines require the use of temporal gates when transformed to fault trees. In many cases, standard Boolean logic is adequate to represent the behaviour and perform a correct analysis; for example, a SM where two events A then B lead to a 'Degraded' state, and two other different events C then D lead to a 'Failed' state. In this case, we are concerned with separate combinations of events, i.e., the two branches have no events in common and each event contributes only to a single end state (and ultimately, contributes to only one top event). Here, a change in the sequence of the events (e.g. D before C instead of C before D) will not lead to a different failure. In the case that D happens first, the system simply stays in the initial state and when C occurs it performs two instantaneous transitions to reach the final failed state; thus the failure behaviour is not sequence-dependent as in the case of the SM in Figure 2. Therefore, in this scenario, Boolean logic is sufficient to model the situation unambiguously (Degraded = A.B and Failed = C.D) and thus the simpler transformation of SMs into fault trees described in [6] is sufficient to obtain an accurate analysis.

The choice between these two approaches is dependent on whether or not the SM has at least one event appearing in more than one path. Typically, if there is at least one event that contributes to the occurrence of more than one system failure, then conversion to temporal fault trees may be needed. It may also be true even if there is an event that is a contributory factor to the occurrence of only one system failure, but as a result of more than one sequence of events. In such cases, an accurate analysis depends upon the correct preservation of the temporal semantics, as different sequences of those shared events — or other events relative to those shared events — may lead to different final states (and thus different system failures). It is also possible for both approaches to be applied in different parts of the system, allowing complex dynamic analysis to be applied only when necessary.

3.3 Automatic generation of Pandora formulae

Informally, normalised graphical notations, e.g. Figure 2 (and Figures 4 to 8), describe SMs of systems, subsystems or components. States are represented by ellipses or circles and events are represented by arrows that join states. Each ellipse or circle contains a textual description of its corresponding state. Similarly, arrows are labelled with textual descriptions of the corresponding events.

A state machine has a finite number of states. It may change state when an event occurs, but at each instant it is in only one state. Definition 1 formally describes a state machine.

Definition 1: ***State Machine (SM)***
A state machine is a quadruple (S, Σ, δ, s_0) where:

- *S is a finite set of states.*
- *Σ is a finite set of events, such that $S \cap \Sigma = \emptyset$.*
- *δ is a partial function: $S \times \Sigma \longrightarrow S$ s.t. for $(u, u') \in S^2$ and $e \in \Sigma$, $u' = \delta(u, e)$ iff e is incident from u to u', and we write it as: $u \xrightarrow{e} u'$.*
- *s_0 is the initial state.*

For example, consider again Figure 2. The SM that formally describes the sensor is defined as follows:

- *S = {Severely Failed, Prematurely Failed, Safely Failed, Omission Detected, Monitoring}.*
- *Σ = {O-A, S fails}. O-S (in the same SM) represents an error propagation — entering the state Severely Failed. It is an event (input deviation) in another SM description.*
- *δ is defined as shown in the SM figure.*
- *s_0 = Monitoring.*

For the purpose of this study, we assume a state machine to be acyclic. A cyclic SM may imply that failures are repairable or repeatable, which is incompatible with the semantics of Pandora. There is, therefore, a finite set of possible paths in the state machine. If π is a path from u to u' $(u, u') \in S^2$, we write it as $u \overset{\pi}{\rightsquigarrow} u'$.

Definition 2: ***Paths set***
Let P be the set of all paths in the SM,
$P = \{\pi \mid u \overset{\pi}{\rightsquigarrow} u', (u, u') \in S^2\}$

We write $u \rightsquigarrow u'$ iff $\exists \pi \in P$ s.t. $u \overset{\pi}{\rightsquigarrow} u'$. In such a case, state u' is said to be reachable from state u. However, if there exists a one-event path from u to u', then u' is said to be immediately reachable from u and we write it as $u \rightarrow u'$ (i.e.,

$u \rightarrow u'$ iff $\exists e \in \Sigma$ s.t. $u \xrightarrow{e} u'$).

We assume that from s_0 we can reach any other state, i.e., $\forall t_{\neq s_0} \in S$ $s_0 \rightsquigarrow t$. Also, $\forall \pi \in P$ $\exists (u, u') \in S^2$ s.t. $u \overset{\pi}{\rightsquigarrow} u'$ and $Seq_\pi = \langle u_0, u_1, \ldots, u_n \rangle$ is the sequence of the states of the path π, where $n = length(\pi)$ is the number of events that label π and the sequence is ordered for backward traversal, i.e., $u_0 = u'$ and $u_n = u$.

A state machine path can be traversed forwards or backwards. The Pandora formula which corresponds to a final state is generated by performing backward traversals of all paths starting from that final state to the initial state. Forward traversals are performed starting from every 'join' state (i.e., with an out-degree strictly greater than 1) to, at worst, all reachable final states, or until the condition for imposing a temporal constraint is satisfied.

Definition 3: ***Forward and backward incidence sets***
For any state $u \in S$, let $\Sigma_{u\blacktriangleright}$ (resp. $\Sigma_{u\blacktriangleleft}$) be the set of events incident from u (resp. incident to u),
$\Sigma_{u\blacktriangleright} = \{e \in \Sigma | \exists u' \in S \text{ s.t. } u \xrightarrow{e} u'\}$
$\Sigma_{u\blacktriangleleft} = \{(e, u') \in \Sigma \times S | \ u' \xrightarrow{e} u\}$

The choice of an event in $\Sigma_{u\blacktriangleright}$ uniquely determines the state that the event is incident to. As for $\Sigma_{u\blacktriangleleft}$, an event incident to u needs to be distinguished from every possible identical event also incident to u, but from a different state. This is done by associating each event with the state from which it is incident, and hence the definition 3 above.

All final states are permanent states — there are no events that lead from a final state to any other state.

Definition 4: ***Set of final states***
Let F be the set of the final states,
$F = \{f \in S | \ \Sigma_{f\blacktriangleright} = \emptyset\}$

There is a corresponding Pandora formula for each final state. Let ϕ_s $s \in F$ be the formula for a final state s. ϕ_s is the disjunction over the paths π (from the initial state s_0 to s) of the conjunction of events that label π. Moreover, any one of the events e that label π, which is incident from a state u with an out-degree of two or more (a state at which paths diverge), is the input event, which must occur first or alone, of a POR gate that associates it with the disjunction of other events e' incident from u to the states u' if, and only if, the subpath of π from u to s shares an event with the paths from u through u' to any state.

Algorithm 1 generates a set Φ of Pandora formulae: $\Phi = \{\phi_s | s \in F\}$ — one formula ϕ_s for each final state s. These expressions can then be analysed by Pandora. For each join state s_{i_j} during a path traversal (line 11 of the Algorithm), and for each event $e'_{\neq e_{i_j}} \in \Sigma_{s_{i_j}\blacktriangleright}$ (line 13), a temporal constraint is imposed (line 16) if, and only if, the two subsets Ω and Σ' (defined thereafter) of Σ share an event ($\Sigma' \cap \Omega \neq \emptyset$). The subsets are such that $\Omega = \{e_{i_j}, e_{i_{(j-1)}}, \ldots, e_{i_1}\}$ and Σ' being initially $\{e'\}$ (e' as specified in lines 13 and 14), but possibly further populated as shown in Algorithm 2 — PopulateAt($t = \delta(s_{i_j}, e')$). This additional population of Σ' can happen only if the state that e' is incident to is not a final state — $t_{\in S} = \delta(s_{i_j}, e')$ is such that $\Sigma_{t\blacktriangleright} \neq \emptyset$. In such a case, the paths from t are traversed forwards as long as Σ' and Ω remain disjoint sets (line 5 and line 8 of Algorithm 2).

Algorithm 1 Φ — *(SM = (S, Σ, δ, s_0))*

1: get P from the state machine
2: get F from S
3: let $\Phi = \emptyset$
4: **for each** $s \in F$ **do**
5: get Ps from P where $Ps = \{\pi \in P | \ s_0 \overset{\pi}{\rightsquigarrow} s\}$
6: let $n = |Ps|$
7: let $\pi_i \in Ps$ $1 \leq i \leq n$ /* $s_0 \overset{\pi_i}{\rightsquigarrow} s$ */
8: let $len_i = length(\pi_i)$ $1 \leq i \leq n$ /* the number of events of π_i */
9: let $Seq_{\pi_i} = \langle s_{i_0}, s_{i_1}, \ldots, s_{i_{len_i}} \rangle$ $s_{i_j} \in S$ $e_{i_j} \in \Sigma$ $1 \leq i \leq n$ $1 \leq j \leq len_i$ s.t. $s_{i_j} \xrightarrow{e_{i_j}} s_{i_{(j-1)}}$ where Seq_{π_i} is ordered from $s_{i_0} = s$ to $s_{i_{len_i}} = s_0$ /* Seq_{π_i} $1 \leq i \leq n$ is the sequence of states of π_i ordered for backward traversal*/
10: let $\phi_s = \bigvee_{1 \leq i \leq n} (\bigwedge_{1 \leq j \leq len_i} e_{i_j})$ /*ϕ_s is the disjunct over paths π of the conjunct of events of π*/
11: **for each** s_{i_j} in Seq_{π_i} $1 \leq i \leq n$ $1 \leq j \leq len_i$ s.t. $|\Sigma_{s_{i_j}\blacktriangleright}| > 1$ **do**
12: let $e = e_{i_j}$ /* $\Sigma_{s_{i_j}\blacktriangleright}$ is the set of events incident from s_{i_j} */
13: **for each** $e' \neq e_{i_j}$ in $\Sigma_{s_{i_j}\blacktriangleright}$ **do**
14: let $t \in S$ s.t. $s_{i_j} \xrightarrow{e'} t$
15: **if** $((e' \in \{e_{i_{(j-1)}}, e_{i_{(j-2)}}, \ldots, e_{i_1}\})$ **or** $(\exists (y \in \Sigma, \pi' \in P, f \in S, v \in S$ in $Seq_{\pi'}$, $w \in S$ in $Seq_{\pi'})$ s.t. $t \overset{\pi'}{\rightsquigarrow} f$, $v \xrightarrow{y} w$ and $y \in \{e_{i_j}, e_{i_{(j-1)}}, \ldots, e_{i_1}\}))$ **then**
16: let $e = e|e'$ /* e' is one of the events that label subpath h of π_i s.t. $s_{i_j} \overset{h}{\rightsquigarrow} s$, or h and π' share an event */
17: **end if**
18: **end for**
19: replace e_{i_j} with e in ϕ_s
20: **end for**
21: let $\Phi = \Phi \cup \{\phi_s\}$
22: **end for**

The conversion algorithm is biased towards increasingly dynamic systems. The best-case complexity of checking the necessity of a temporal order is $O(n)$ and the worst case is $O(n^2)$, n being the number of paths from the initial state to the final states in the SM. The best case is a SM where for every divergent path, there exists a sharable event[1] that is incident to the immediately reachable state from the join state at which the path diverges. In such a situation, the price to pay for each join state j is an $O(m^2)$ operation of temporal order enforcement, and where m is the out-degree of j.

The worst case is a SM where for every divergent path, either there are no sharable events or there exists only one sharable event — immediately before the last reachable final state. A SM where there are no sharable events at all (e.g. SM of a static system) is also a worst-case scenario.

To show the quadratic growth of the conversion algorithm with the number of full paths, we define a marking function that marks every state with the number of paths to the final states (these latter states are special cases which are marked

[1] An event which causes the condition for imposing a temporal constraint to be satisfied.

Algorithm 2 *PopulateAt(S: t)*

```
1: let e∈Σ_{t▶}
2: let Σ' = Σ' ∪ {e}
3: let t'_{∈S} = δ(t, e)
4: let Σ_{t▶} = Σ_{t▶} \ {e}
5: if ((Σ' ∩ Ω = ∅) and (Σ_{t'▶} ≠ ∅)) then
6:    PopulateAt(t')
7: end if
8: if ((Σ' ∩ Ω = ∅) and (Σ_{t▶} ≠ ∅)) then
9:    PopulateAt(t)
10: end if
```

with 1). Let $M : S \rightarrow \mathbb{N}^+$ be that marking function. The number of paths from a state u to the final states is the sum, for all events incident from u to its successors (i.e., the immediately reachable states from u), of the number of paths from these successors to the final states (i.e., their corresponding markings), and hence:

$$M(\mathrm{u}) = \begin{cases} 1 & \text{if } u \text{ is a final state} \\ \sum_{e \in \Sigma_{u\blacktriangleright}} M(\delta(u, e)) & \text{otherwise} \end{cases}$$

This algorithm will add the marking of each successor v as many times as there are events that are incident from u to v. Apart from the final states, every state u will be marked such that $M(\mathrm{u}) = |\mathrm{P}^u|$, where $\mathrm{P}^u = \{\pi \in \mathrm{P} |\ \mathrm{u} \overset{\pi}{\rightsquigarrow} \mathrm{f}, \mathrm{f} \in \mathrm{F}\}$. For example, marking the states of the SM of D (Figure 7) gives $M(\mathrm{ON}) = M(\delta(\mathrm{ON}, \text{O-C})) + M(\delta(\mathrm{ON}, \text{O-B Severe})) + M(\delta(\mathrm{ON}, \text{D fails}))$, totaling 3 full paths from the initial state 'ON'. This marking conforms to the backward traversals that are performed from each of the final states (line 9 of Algorithm 1).

Let N be the total number of paths (from the initial state to the final states). During each backward traversal starting from a final state s, at every visited state u that is a join state (i.e., line 11 of Algorithm 1), at worst, all remaining paths to all possible final states will be exhaustively traversed forwards by Algorithm 2. This results in a total number $M(\mathrm{u})$ of traversed paths. This number is preserved along the backward traversal at every state that is not a join state until another join state gets visited, ultimately the initial state s_0 with $M(\mathrm{s}_0)$ equals to N. This will be repeated Ns times for the final state s, where Ns is the total number of paths from s_0 to s. Thus, for all final states we have a total number of traversed paths which is equal to $\mathrm{N} \times \sum_{\mathrm{s} \in \mathrm{F}} \mathrm{Ns}$, i.e., N^2. Note that to determine each Ns $\mathrm{s} \in \mathrm{F}$, we can use another version of the marking function M, which marks the initial state with 1. Thereafter, the number of paths reaching a state u is the sum, for all elements in $\Sigma_{u\blacktriangleleft}$, of the number of paths that reach each predecessor u' (u'→u).

4. COMPOSITIONAL SYNTHESIS OF TFTs AND TEMPORAL ANALYSIS

4.1 The approach

Assuming a system with several components, each modelled by its own SM, it is helpful to be able to construct the TFTs for the analysis in a compositional manner, producing TFTs for each component at a time and combining them to produce the final, system-wide TFT (or set of TFTs). This helps to minimise the expense of the transformations and also simplifies the task of the system designer, allowing them to focus on modelling local failures in each component.

The first step in the compositional approach is to generate a small TFT (in the form of a Pandora expression) for each final state of each component SM, using the algorithm presented earlier. If distinct final states of a component's SM represent the same output deviation — such as an omission of output from that particular component — then the approach merges their TFTs into a disjunction (since either tree can lead to this failure). The resulting TFT expressions can contain both symbols which represent basic events (e.g. internal failures of the component) as well as non-atomic events (i.e., faults caused by failures in some other components). These non-atomic events are input deviations of the component for which the TFT has been generated and typically correspond to output deviations of connected source components. The newly generated TFTs for each component can then be minimised if appropriate to obtain a simplified intermediate form; this helps to remove any redundancies or complexities as early in the process as possible.

The next step is to synthesise system fault trees from the smaller component fault trees by combining them, joining the inputs of one component fault tree to the outputs of the next and so forth. This is done by working backwards, starting with the Pandora expressions of the system failures. When it finds an input deviation, it looks for the corresponding local TFT expression for the matching output deviation in the source component(s) and merges them into a bigger fault tree expression by replacing the input deviation event with the output deviation expression. At each stage, the fault tree expression is minimised wherever possible to remove any new redundancies or contradictions etc. When no more non-atomic events remain, the expression cannot be expanded any further as all possible substitutions have taken place, and the complete fault tree has been synthesised for that particular system failure. At this point, a final analysis takes place to obtain the minimal cut sequences (or minimal cut sets, if it contains no temporal logic) and thus determines the sequences or combinations of failures that lead to that system failure.

4.2 Case study - Introduction

To better illustrate how our compositional approach works, we use a Generic Triple-module Redundant (GTR) system, as shown in Figure 3. Components A, B and C are abstract representations of any kind of input, control or actuating device, arranged in a redundant series, with A as the primary component, B as the secondary, and C as tertiary. Two monitoring sensors, S1 and S2, detect omission of output from A and B respectively and activate the next backup in the series. D is simply an abstraction of the output of the GTR and 'I' represents the input to the system.

Performing standard FTA on the system will result in the list of MCS below [18]:

1. Omission of input at I.
2. All of A, B, and C fail.
3. Both A and S1 fail (B will not be activated).
4. All of A, B, and S2 fail (C is not activated).
5. Failure of D.

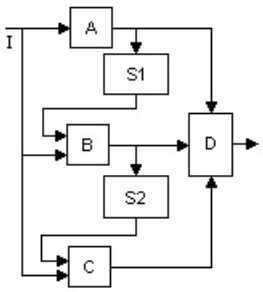

Figure 3: GTR system

At first glance, all appear to be correct. If there is no input to the system, then it cannot operate; similarly, if the output component of the system fails, then the system cannot function. If all three main components (A, B, and C) fail, then the system will likewise fail as well. #3 and #4 are more complex, but detail situations where the monitoring sensors themselves fail, and are thus unable to detect a failure of the monitored component and activate the next backup, leading to premature system failure.

However, like the system of Figure 1, the GTR system exhibits dynamic behaviour: its true failure behaviour depends on the chronology of events. The system can function in any of three modes — with A active, B active, or C active — and the transition between those modes is triggered by omission failures detected by the monitors S1 and S2. Different sequences of failure events can lead to the system failing in different ways, and not all of them are correctly represented by the results.

For example, assume that B fails first, then A fails second; in this case, sensor S1 will not be able to activate B when it detects omission of output from A, and sensor S2 will not activate C because B was never activated — it can only detect an omission of output once its monitored component is activated. Thus the sequence "B fails before A fails" will cause the system to fail regardless of the status of C and S2. This means that MCS #2 and #4 are unduly optimistic — in certain cases, the system will fail without all of the events in those MCS having to occur.

Similarly, S1 failing before A means that component B will not be activated upon omission of output from A, since the monitor is not operational and thus unable to detect the omission. Again, this means that the system can fail irrespective of the state of S2 and C. However, if S1 fails after A, then it has no effect on the system: it has already served its purpose and activated B, so unless there are further failures elsewhere, the system will continue to operate. Thus MCS #3 can be unnecessarily pessimistic in this case. The same is also true of the relationship between B and S2.

Clearly, standard combinatorial fault trees do not always produce accurate results for even simple dynamic systems like this one. In this example, we have seen that although FTA suggests a failure of all of A, B, and C is necessary to cause system failure, a failure of B before A is sufficient, giving a false sense of security; this information, had it been known, may have resulted in a different system design, e.g. a triple voter or some other different redundancy architecture.

To remedy this problem, we show how we can generate the TFTs from the SMs of the individual components. Thereafter, we demonstrate our approach by building the system

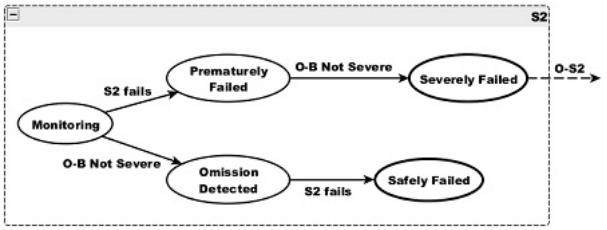

Figure 4: SM of S2

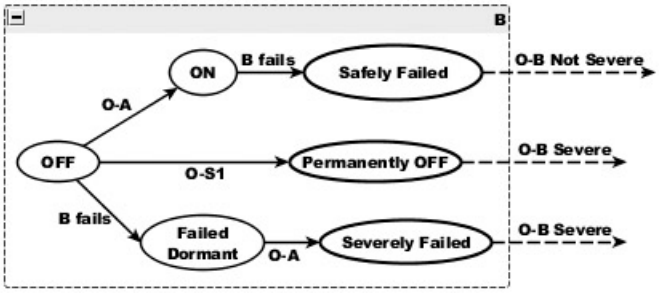

Figure 5: SM of B

TFT (in this case there is only one – an omission of output from D, i.e., O-D) and then performing a temporal qualitative analysis on it.

4.3 Case study - Modelling of component SMs

First we model the failure behaviour of each component using SMs. Sensors S1 and S2 have similar jobs (which is to activate the next backup if omission of output is detected from the monitored component). There is therefore potential for reuse of the sensor SM from Figure 2, with 'O-A' replaced by an input failure from A or B and 'S fails' becoming 'S1' or 'S2 fails' as appropriate. The state machine for S2 is shown in Figure 4; the SM for S1 is identical to Figure 2 except that 'S1 fails' and 'O-S1' replace 'S fails' and 'O-S'.

However, for the SM of S2 (shown in Figure 4), we need to distinguish between two types of omission of output from B. 'O-B Not Severe' is a detectable omission of output from B and occurs when B has been activated by S1 upon failure of A and then subsequently fails; it is the effect of entering state 'Safely Failed' in the SM of B (see Figure 5) and allows S2 to activate C, assuming S2 itself has not yet failed. The other kind of omission is 'O-B Severe', an undetectable omission of B caused either by B failing dormant or S1 failing prematurely; either case means that S1 is unable to activate B upon failure of A. This omission is the effect of B entering the 'Permanently OFF' or 'Severely Failed' states, as seen in Figure 5.

As can be seen from Figure 7, 'O-B Severe' will lead the system to a total failure and an omission of output from D (O-D). This in turn alters the state of the GTR system to 'Failed' (i.e., complete failure of the system) as shown in Figure 8. The SM of D also shows an additional cause for O-D: the basic event 'D fails'. This is a single point of failure for the system.

The only other single point of failure is omission of input to the system (O-I), which affects the whole GTR; this is represented at the system level rather than repeated in each

component, as can be seen in Figure 8. Thus, the SM of component A (not shown, due to its triviality) has simply one path from 'ON' to 'Failed', with the effect of omission of output from A (O-A). The state transition of A is assumed to be triggered only by a basic event of the component (e.g. an internal failure 'A fails'), since a lack of input is represented by O-I at the system level.

For component C (SM of Figure 6), there are two final states, both with omission of output from C (O-C) as common effect. The state 'Permanently OFF' means that C will never be activated and is caused by O-S2 (i.e. S2 has prematurely failed and both A and B have failed in sequence). The other final state is a state where A and B have failed in sequence and where C has also failed (caused by the event 'C fails'). Omission from C is the third cause of failure for output component D, leading to O-D and system failure.

Note that A and D are each initially in state 'ON' — i.e. the GTR system is initially working with its primary component, and the output is being delivered by the system. Backup components have their initial states set to 'OFF' and sensors are initially set to 'Monitoring'.

4.4 Case study - Synthesis and analysis of TFTs

At this stage, we are ready to generate, for each component, a set of fault trees — one TFT per final state. Thereafter, we merge the fault trees that correspond to the same output deviation into one single fault tree. For the sake of clarity, any component symbol (e.g. A) abbreviates the failure (e.g. 'A fails') of the corresponding component.

The TFT of component A is the simplest: O-A = A (i.e., omission of A is caused by failure of A). The SM of component B, however, has three final states; entering two of these states ('Permanently OFF' and 'Severely Failed') will cause the same output deviation (O-B Severe), and thus the TFT for O-B Severe is a disjunction of the generated TFTs for those two states. If we apply the algorithm on the SM of B we get:

Permanently OFF = O-S1
Severely Failed = O-A.(B|O-A) (equivalent to B<O-A)
Safely Failed = B.(O-A|B) (equivalent to O-A<B)

Therefore, O-B Severe = O-S1 + O-A.B|O-A, and O-B Not Severe = B.O-A|B.

We can perform some initial minimisation at this stage by applying the temporal law (A|B).B ⇔ A<B; this results in the following:

O-B Severe = O-S1 + B<O-A
O-B Not Severe = O-A<B

For S1 and S2, we will get O-S1 = Severely Failed (of S1) = O-A.S1|O-A and O-S2 = Severely Failed (of S2) = (O-B Not Severe).S2|(O-B Not Severe). The other failure states (labelled 'Safely Failed') have their own fault trees, but they will not be synthesised since they do not correspond to output omissions (these are still considered as safe states). Again we can apply the law (A|B).B ⇔ A<B to obtain:

O-S1 = S1<O-A
O-S2 = S2<O-B Not Severe

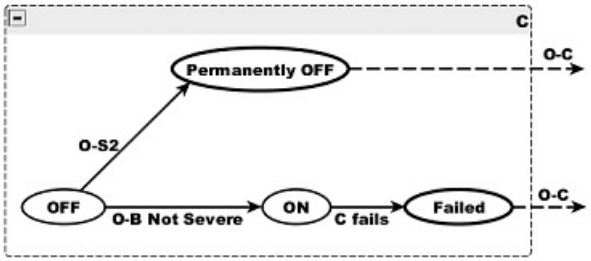

Figure 6: SM of C

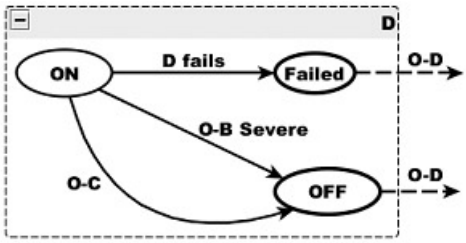

Figure 7: SM of D

If we apply the algorithm on the SM of component C, we will get the below result:

Permanently OFF = O-S2
Failed = C.(O-B Not Severe)

and thus *O-C = O-S2 + C.(O-B Not Severe).*

At this stage, no reduction is possible here.

For component D, we have:

O-D = D + O-C + O-B Severe

since all final states trigger the same omission of output.

Finally, the the failure state of the whole GTR system is represented as follows:

Failed = O-I + O-D

The final step of the synthesis process is to join these expressions together, linking input deviations to output deviations. For example, O-A has only one cause (A), so any instance of O-A can be replaced by A, e.g. in O-S1, which becomes S1<A; similarly, we can now replace O-S1 with this expression, e.g. O-B Severe = S1<A + B<A. The full expression representing the failure of the GTR system is therefore as follows:

Failed = D + S2<(A<B) + C.(A<B) + S1<A + B<A + O-I

Now that we have the full expression, final minimisation can take place. In this example, most of the possible reduction has already taken place, but we can expand the parentheses in the expression S2<(A<B); when we have nested PAND gates like this, the brackets can be removed by considering all the possible sequences it represents. Here, because there is no order specified between S2 and A, there

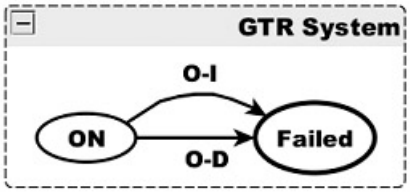

Figure 8: SM of the GTR

are two new sequences: S2<A<B and A<S2<B. This gives us the list of seven final Minimal Cut Sequences (MCSQs), which more concisely describe how the system can fail:

1. Failure of output block D.
2. S2 failing before A, and A failing before B.
3. A failing before S2, and S2 failing before B.
4. All three main components A, B and C fail, with A failing before B.
5. S1 failing dormant before it detects failure of A.
6. B failing dormant before A fails, so C never activates.
7. Omission of system input.

These are more meaningful and subtle than the original five minimal cut sets provided by standard combinatorial FTA. In particular, the effects of dormant failures — like B failing before A, or a sensor failing before activating its backup — can now be taken into account.

4.5 Comparison to direct Pandora analysis of the TFT

The minimisation of the cut sequences — which is typically the most expensive part of the analysis process — is simplified considerably by the manner in which the expressions are synthesised from component SMs instead of modelling the system entirely using temporal logic. If the system was originally modelled as a TFT instead of using component SMs, we may have arrived at a fault tree expression such as this:

Failed = D + S1<(A+O-I) + (B + O-I)<(A+O-I) + (O-I + (S2<((A + O-I)<B + (A + O-I)<O-I)) + C . ((A + O-I)<B + (A + O-I)<O-I))

This has many more nested expressions and takes considerably more time and effort to minimise. For example, subexpression (B + O-I)<(A + O-I) must be split into two new expressions: B<(A + O-I) and O-I<(A + O-I). The former can be reduced to obtain B<A|O-I and B<O-I|A, while the latter contains contradictions that would have been eliminated much earlier during compositional process. Instead, they must be eliminated now:

1. *O-I<(A+O-I)*
2. *(O-I|A).(O-I|O-I).(A+O-I)*
3. *(O-I|A).0.(A+O-I)*
4. *0*

(S2<((A + O-I)<B + (A + O-I)<O-I) is the most complicated sub-expression and correspondingly the most difficult to reduce. Again, with a compositional approach, much of this complexity would have been simplified during earlier stages; instead, because it is only minimised at the end, this complex expression eventually yields 21 further cut sequences:

O-I<S2<A<B + S2<O-I<A<B + S2<A<O-I<B + S2<A<B|O-I + O-I<A<S2<B + A<S2<O-I<B + A<S2<B|O-I + A<S2<O-I<B + S2<A<O-I<B + S2<O-I<A<B + S2<O-I<B|A + O-I<A<S2<B + O-I<S2<A<B + O-I<S2<B|A + B<S2<A<O-I + S2<B<A<O-I + S2<A <B<O-I + S2<A<O-I|B + B<A<S2<O-I + A<S2<B<O-I + A<S2<O-I|B

The first stage of the analysis ultimately results in a set of 30 cut sequences, but not all of these are minimal. Thus far, all reduction has taken place within the cut sequences, i.e. within a single conjunction or temporal sequence. The next step is to minimise cut sequences against each other. For example, the rule A|B+B ⇔ A+B can be applied to several cut sequences (e.g. B<A|O-I becomes B<A) and the absorption laws A<B + A ⇔ A and A<B + B ⇔ B can be applied to many others (e.g. B<O-I|A can be eliminated, as can e.g. O-I<S2<A<B etc). This gives us a much reduced final list of minimal cut sequences:

1. *D*
2. *S1< A*
3. *B< A*
4. *O-I*
5. *S2<A<B*
6. *A<S2<B*
7. *C.A<B*

These seven MSCQs are the same as before, but the steps needed to reach them are much more complex, requiring far more time and effort. The compositional approach, building and analysing TFTs from component SMs, is simpler to understand from both a modelling and analysis perspective and considerably more scalable than using TFTs directly. Furthermore, the scaling benefits of using compositional SMs for the initial dynamic modelling when compared to using temporal FTA directly — in terms of both understandability and ease of analysis — increase with the size of the system being studied.

5. CONCLUSION

Dynamic systems pose significant challenges for traditional safety analysis techniques like FTA, which are typically static in nature and cannot easily represent the dynamic failure behaviour exhibited by such systems. Temporal FTA approaches exist that can help solve these issues, but suffer from a number of drawbacks: some, such as Dynamic Fault Trees and CSDM, are not primarily designed for qualitative analysis, which limits the situations in which they can be applied; others, like Pandora, are suitable for qualitative analysis but are generally more expensive and less scalable than normal FTA. In all cases, the FTA techniques require that the system be modelled using fault trees, which can often be less intuitive and understandable for dynamic systems than for purely static systems.

To better model the dynamic failure behaviour, state machines are often used, but these typically require conversion to some other format (like Petri nets or fault trees) before they can be analysed. Both conversions have their problems: Petri nets are less suitable for qualitative analysis, which is

important during early stages of the design if problems are to be identified and remedied cheaply, while existing SM-to-FT methods focus on converting them to standard combinatorial fault trees, which can result in errors in situations where different sequences of the same events have different outcomes. Instead, a method of converting SMs to a format suitable for dynamic qualitative analysis is required.

In this paper, we solve this problem by presenting a different approach which converts compositional state machines into Pandora temporal fault trees. This has the dual benefits of allowing more detailed analysis of different sequences of events, thus better capturing the dynamic behaviour represented by the original state machines, and also helping to reduce the cost of performing temporal FTA by producing smaller, more manageable temporal fault tree expressions via the compositionality, resulting in a more scalable approach overall.

We demonstrated this technique on a simple generic redundant system and showed how the SMs can be converted into temporal expressions and subsequently analysed to produce useful qualitative information about the dynamic failure behaviour of the system. By using compositionality, the scalability is improved further, since each individual SM is relatively small. As demonstrated by the case study, this offers significant benefits over using TFTs alone. We hope that this approach can be developed further and become automated as part of the HiP-HOPS safety analysis tool framework. We also aim to extend it further and potentially make it compatible with other SM-based modelling approaches, such as interfacing with OSATE[2] (an Open Source AADL Tool Environment), allowing Pandora TFTs to be generated from AADL state machines.

In summary, performing meaningful qualitative safety analysis on dynamic systems means using both dynamic modelling and dynamic analysis approaches; by converting SMs to temporal FTs, it becomes possible to do both and thus it is hoped that this technique can contribute to an improvement in the overall safety of increasingly complex modern safety-critical systems.

6. ACKNOWLEDGEMENT

This study was partly funded by the FP7 project MAENAD (Grant 260057).

7. REFERENCES

[1] Y. Papadopoulos. Safety-Directed System Monitoring Using Safety Cases. PhD Thesis. The University of York, Department of Computer Science, 2000.

[2] A. Rauzy. Mode automata and their compilation into fault trees. *Reliability Engineering and System Safety*, 78(1), pp. 1-21, 2002.

[3] M. Bozzano, and A. Villafiorita. Improving System Reliability via Model Checking: the FSAP /NuSMV-SA Safety Analysis Platform. In *Proceedings of SAFECOMP 2003*, Edinburgh, Scotland, United Kingdom, 23-26 September, pp. 49-62, 2003.

[4] P. H. Feiler, and A. E. Rugina. Dependability Modeling with the Architecture Analysis and Design Language (AADL). Carnegie Mellon Software Engineering Institute, N° CMU/SEI-2007-TN-043, 2007.

[2] http://www.aadl.info/aadl/currentsite/tool/osate.html

[5] A. Rugina. Dependability Modeling and Evaluation – From AADL to Stochastic Petri Nets, PhD Thesis. Institut National Polytechnique de Toulouse - INPT, France, 2007.

[6] A. Joshi, S. Vestal, and P. Binns. Automatic Generation of Static Fault Trees from AADL Models. In *DSN Workshop on Architecting Dependable Systems*, Edinburgh, Scotland – UK, 2007.

[7] R. Bernard. Multi-system safety analyses, PhD Thesis. N° d'ordre: 3885. Université Bordeaux I, École Doctorale de Mathématiques et Informatique, France, 2009.

[8] N. Mahmud, Y. Papadopoulos, and M. Walker. A Translation of State Machines to Temporal Fault Trees. In *Proceedings of 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, Chicago, Illinois, June 28-July 1, pp. 45-51, 2010.

[9] J. B. Fussel, E. F. Aber, and R. G. Rahl. On the quantitative analysis of Priority-AND failure logic. *IEEE Transactions on Reliability*, Volume R-25/5, pp 324-326, 1976.

[10] W. E. Vesely, M. Stamatelatos, J. B. Dugan, J. Fragola, J. Minarick , and J. Railsback. Fault Tree Handbook with Aerospace Applications. NASA Office of Safety and Mission Assurance, USA, 2002.

[11] R. Gulati, and J.B. Dugan. A Modular Approach for Analyzing Static and Dynamic Fault Trees. In *Proceedings of the Annual Reliability and Maintainability Symposium*, Philadelphia, Pennsylvania, 13-16 pp 57-63, Jan 1997.

[12] S. Amari, G. Dill, and E. Howard. A new approach to solve dynamic fault trees. In *Proceedings of the Annual Reliability and Maintainability Symposium* ISBN 0-7803-7717-6, pp 374-379. 2003.

[13] K. Durga Rao, V. Gopika, V.V.S Sanyasi Rao, H.S. Kushwaha, A.K. Verma, and A. Srividya. Dynamic Fault Tree analysis using Monte Carlo simulation in probabilistic safety assessment. *Reliability Engineering and System Safety* Vol 94, 872-883. 2009.

[14] G. Merle, J-M Roussel, and J-J Lesage. Algebraic determination of the structure function of Dynamic Fault Trees. *Reliability Engineering and System Safety* Vol 96:2 , pp 267-277. DOI: 10.1016/j.ress.2010.10.001. 2011.

[15] J. Gorski, and A. Wardzinski. Deriving Real-Time requirements for software from safety analysis. In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*, IEEE CS Press, pp 9-14. 1996.

[16] J.Gorski and A Wardzinski. Timing aspects of fault tree analysis of safety critical systems. Proceedings of the Safety Critical Systems Symposium, Brighton, UK, Feb 1997. pp 231-244.

[17] M. Walker, L. Bottaci, and Y. Papadopoulos. Compositional Temporal Fault Tree Analysis. In F. Saglietti and N. Oster, editors, *Computer Safety, Reliability, and Security – SAFECOMP'07, Lecture Notes in Computer Science 4680*, pp. 105-119. Springer ISBN 978-3-540-7, 2007.

[18] M. Walker. Pandora – A Logic for the Qualitative Analysis of Temporal Fault Trees, PhD Thesis. University of Hull, UK, 2009.