An overview of Fault Tree Analysis and its application in model based dependability analysis

Sohag Kabir[*]
School of Engineering and Computer Science, University of Hull, Hull, HU6 7RX, UK

Abstract — Fault Tree Analysis (FTA) is a well-established and well-understood technique, widely used for dependability evaluation of a wide range of systems. Although many extensions of fault trees have been proposed, they suffer from a variety of shortcomings. In particular, even where software tool support exists, these analyses require a lot of manual effort. Over the past two decades, research has focused on simplifying dependability analysis by looking at how we can synthesise dependability information from system models automatically. This has led to the field of model-based dependability analysis (MBDA). Different tools and techniques have been developed as part of MBDA to automate the generation of dependability analysis artefacts such as fault trees. Firstly, this paper reviews the standard fault tree with its limitations. Secondly, different extensions of standard fault trees are reviewed. Thirdly, this paper reviews a number of prominent MBDA techniques where fault trees are used as a means for system dependability analysis and provides an insight into their working mechanism, applicability, strengths and challenges. Finally, the future outlook for MBDA is outlined, which includes the prospect of developing expert and intelligent systems for dependability analysis of complex open systems under the conditions of uncertainty.

1. Introduction

Safety critical systems are extensively used in many industries, including the aerospace, automotive, medical, and energy sectors. Systems that fall into this category range from airbags in cars to propulsion systems on spacecraft; however, they all share a common property — their failure has the potential to cause catastrophic effects on human life as well as the environment. For this reason, it is expected that safety critical systems possess a high level of dependability. Dependability is the capability of avoiding failures that are more frequent and more severe than is acceptable, and thus dependability assessment should be carried out early in the design phase to avoid unacceptable costs in terms of loss of life, environmental damage, and loss of resources by identifying and rectifying potential hazards as soon as possible. The dependability of a system includes, but is not limited to the following characteristics: safety, reliability, and maintainability.

There are many widely used classical safety assessment methods available to assist safety analysts in performing dependability analysis of systems. One such widely used method is Failure Modes Effects and Criticality Analysis (FMECA). FMECA was initially specified in US Military Procedure MIL-P-1629 and then updated in MIL-STD-1629A (US Department of Defense, 1980). It is an inductive analysis method that considers all possible combinations of effects of a single component failure mode(s). This method also provides ways to perform probabilistic analysis to determine criticality of failure modes.

The Fault Tree Analysis (FTA) (Vesely, Goldberg, Roberts, & Haasl, 1981) is another well-established and well-understood technique, widely used to determine system dependability. In fault trees, the logical connections between faults and their causes are represented graphically. FTA is deductive in nature meaning that the analysis starts with a *top event* (a system failure) and works backwards from the top of the tree towards the leaves of the tree to determine the root causes of the *top event*. The results of the analysis show how different components failures or certain environmental conditions can combine together to cause the system failure. After construction of a fault tree, the analyses are carried out in two levels: a qualitative level and a quantitative level. Qualitative analysis is usually performed by reducing fault trees to minimal cut sets (MCSs), which are a disjoint sum of products consisting of the smallest combinations of basic events that are necessary and sufficient to cause the top event.

In quantitative analysis, the probability of the occurrence of the top event and other quantitative reliability indexes such as importance measures are mathematically calculated, given the failure rate or probability of individual system component. The results of quantitative analysis give analysts an indication about system reliability and also help to determine which components or parts of the system are more critical so analysts can put more emphasis on the critical components or parts by taking necessary steps, e.g., including redundant components in the system model. The usual

[*] Corresponding author. Email: s.kabir@hull.ac.uk

quantification methods for classical static fault trees do not consider uncertainty in failure data. As the outcome of quantitative analysis is entirely dependent on the precision of the numerical data used in the analysis, if uncertainties are left unresolved then there is a chance of producing misleading results. Different methodologies, mainly based on fuzzy numbers, have been proposed to tackle the issue of uncertain failure data in FTA.

The standard fault tree (SFT) can only evaluate safety and reliability of static systems. Static systems are those which only experience a single mode of operation throughout the duration of their lifetimes, and thus exhibit constant nominal and failure behaviours. However, modern large-scale and complex systems can operate in multiple phases, e.g. an aircraft can operate in take-off, flight, and landing modes. One important characteristic of such systems is their dynamic behaviour, i.e., the behaviour of the system (both nominal and potential failure behaviour) can change according to what state or mode of operation the system is in. Dynamic system behaviour leads to a variety of dynamic failure characteristics such as functional dependent events and priorities of failure events. Although SFTs are widely used for dependability analysis, they are unable to capture dynamic failure behaviour of a system.

Dynamic dependability assessment overcomes many of the limitations of the static dependability analysis by allowing the dependability assessment of dynamic systems. It can capture system behaviour for multiple states and can model many possible interactions between system components and variables. In addition to that, it can capture time- or sequence-dependent behaviour of systems. To facilitate dynamic dependability analysis, the SFTs have been extended in different ways such as dynamic fault trees (DFTs) (Dugan, Bavuso, & Boyd, 1992), state-event fault trees (Kaiser, Gramlich, & Förster, 2007), and Stochastic Hybrid Fault Tree Automaton (SHyFTA) (Chiacchio et al., 2016) etc. DFT is the most widely used dynamic extensions of the SFT and it can capture sequence dependent behaviour, behaviour of functionally dependent components and also the priorities of the events. SHyFTA is a recent approach that combines DFT and the Stochastic Hybrid Automaton (Aubry & Brînzei, 2015; Castaneda, Aubry, & Brînzei, 2011) techniques to perform dynamic reliability assessment.

FTA is primarily a manual process and often performed on informal system models. As the system design evolves, these informal models could rapidly become outdated, which has the potential to make the dependability assessment process inconsistence and incomplete. Over the past two decades, research has focused on simplifying dependability analysis by looking at how we can synthesise dependability information from system models automatically. This has led to the field of Model-Based Dependability Analysis (MBDA) (Joshi, Heimdahl, Miller, & Whalen, 2006). Several tools and techniques such as Hierarchically Performed Hazard Origin & Propagation Studies (HiP-HOPS) (Papadopoulos & Mcdermid, 1999), AADL (Feiler, Lewis, & Vestal, 2006) , and AltaRica (Arnold, Point, Griffault, & Rauzy, 2000) etc. have been developed as part of MBDA. Many of these techniques use fault tree analysis as their primary means of system dependability analysis and automate the fault tree generation process.

A survey on standard fault tree analysis and its extensions is represented in Ruijters & Stoelinga (2015). This survey covered technical details of different types of fault trees and their analyses (both qualitative and quantitative) approaches. A literature review on different model based dependability analysis approaches is available in (Aizpurua & Muxika, 2013; Sharvia, Kabir, Walker, & Papadopoulos, 2015). As described in these reviews, many of the MBDA approaches use fault tree analysis as their primary means of analysis and automate the fault tree generation process from system models. In this paper, at first, I review the standard FTA and describe the limitations of this approach with an example. Afterwards, I review different extensions of the standard fault tree. Finally, different model based dependability analysis approaches where fault trees are used as an analysis technique are reviewed and the concepts of the application of FTA in these approaches are discussed with examples. In doing this, I have reviewed more than 200 papers on fault tree analysis, extensions of fault trees and model-based dependability analysis concepts.

I have used different bibliographical research tools such as Google Scholar (https://scholar.google.co.uk/), ScienceDirect (http://www.sciencedirect.com/), IEEEXplore (http://ieeexplore.ieee.org/), SpringerLink (http://link.springer.com/), Web of Science (https://webofknowledge.com/), ACM Digital Library (http://dl.acm.org/) , and Scopus (https://www.scopus.com/), to obtain the relevant articles. Although an extensive effort has been made to find all the relevant articles, no explicit guarantee can be given that this paper has found every relevant paper.

The remainder of this paper is organised as follow: Section 2 reviews the classical fault tree analysis technique and describes the limitation of this technique. Section 3 presents a bibliographical review of different extensions of the standard FTA. The concept of model based dependability analysis, different MBDA techniques, and the application of FTA in MBDA are reviewed in section 4. Section 5 presents a thorough discussion and future outlook for MBDA. Finally, the concluding remarks are presented in section 6.

2. Standard Fault Trees

FTA was invented in 1961 in Bell Laboratories by H.A. Watson, with the support of M. A. Mearns. The intention behind this invention was to help in the design of US Air Force's Minuteman missile system. The approach was successfully used by David Haasl from the Boeing Company to analyse the whole system. Several papers on fault tree analysis were presented at the first System Safety Conference in 1965 (Ericson, 1999). After the creation of FTA, it has been used in variety of fields, including but not limited to: automotive, aerospace, and nuclear industries (Walker & Papadopoulos, 2009; Kabir, Azad, Walker, & Gheraibia, 2015). The Fault Tree Handbook (Vesely et al., 2002) provides a broad introduction to standard fault trees.

2.1 Fault Tree Symbology

Fault tree consists of three types of nodes: events, gates and transfer symbols. Symbols used in SFTs to represent different events are shown in Fig. 1.

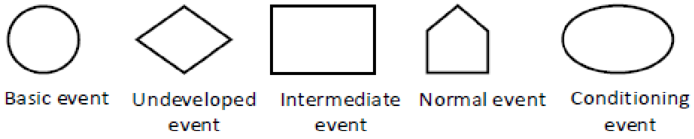


Fig 1. Fault Tree Event Symbols

A basic event is an initiating or basic fault that does not require any further development or expansion and is graphically represented by a circle. Basic events are represented as leaf nodes in the fault tree and they combine together to cause intermediate events. To facilitate quantitative analysis basic events are usually given failure rates and/or repair rates. In the qualitative analysis, cut sets are the combination of different basic events.

An intermediate event is a fault that is caused by the logical combinations of other events occurring further down the tree. As intermediate events are caused by other events, they are almost always a type of logical gate. An undeveloped event is an event whose contributions are not considered in the analysis, either because it is considered as unnecessary, or because insufficient information is available. It is graphically represented by a diamond. A conditioning event does not necessarily represent a fault, it serves as a special condition or constraint for certain types of gates. An ellipse is used to represent a conditioning event. A normal event does not represent any fault and it is part of the nominal behaviour of the system. Normal events are represented by a house symbol.

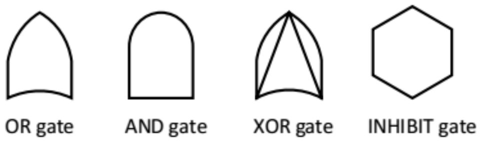


Fig 2. Fault Tree Logic Gate Symbols

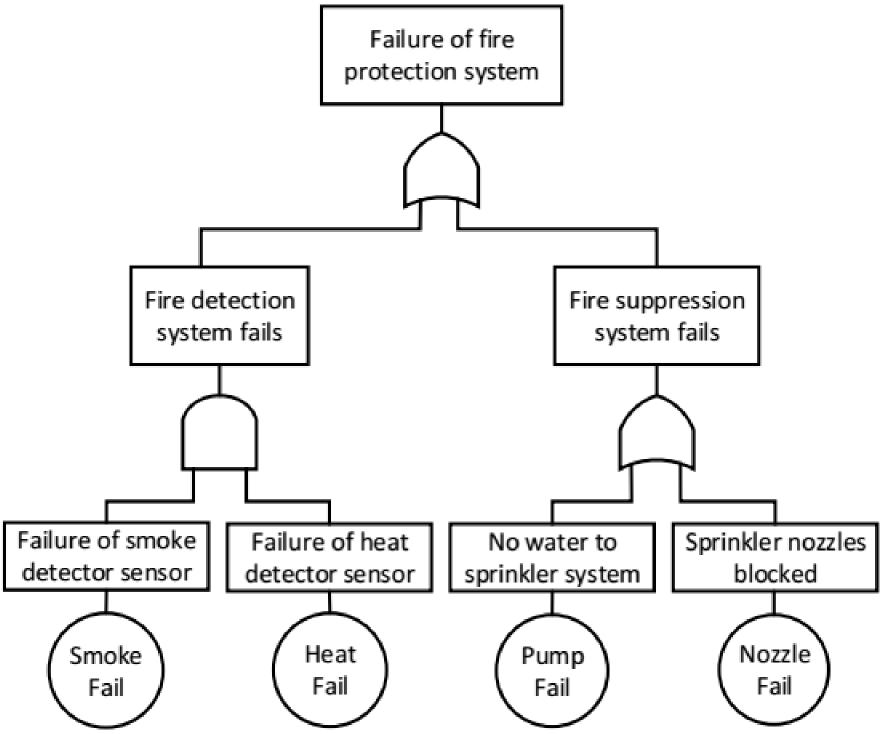

Failure of fire protection system
Fire detection system fails
Fire suppression system fails
Failure of smoke detector sensor
Failure of heat detector sensor
No water to sprinkler system
Sprinkler nozzles blocked
Smoke Fail
Heat Fail
Pump Fail
Nozzle Fail

Fig 3. Example of a standard fault tree (Andrews, 1998)

Symbols used in SFTs to represent different logic gates are shown in Fig. 2. The OR gate is used to show a scenario when the output event occurs if at least one of the input events occur. There is no restriction on the number input events to an OR gate. Inputs to an OR gate are often restatements of the output, i.e., an OR gate does not necessarily represent a causal relationship between its inputs and outputs. The output of an AND gate is true if all of its input events are true. For example, a fire detection system can fail if both smoke detector unit and heat detector unit fail but not by the failure of just one unit. Similar to the OR gate there may be any number of input events to an AND gate but in contrast to the OR gate, the AND gate usually represents a causal relationship between its inputs and outputs. The XOR gate is true if one and only one of its input events is true. This gate is a special case of the OR gate and in most fault tree analysis it is considered as a two-input gate where the output is true if only one of the inputs is true but not two. The INHIBIT gate is a special case of the AND gate and it produces an output when its only input event is true in the presence of a conditioning event. An example of a typical fault tree is shown in Fig. 3.

If a fault tree is too big to fit in a single page, then transfer events are used to spread the fault tree in multiple pages. A transfer In symbol specifies that the tree is developed further and the branch corresponds to this transfer In symbol is displayed in another page at the corresponding Transfer Out symbol. A transfer Out symbol indicates that this branch of tree corresponds to a Transfer In symbol defined earlier and it must be attached with its corresponding transfer In symbol.

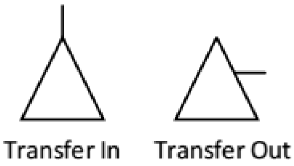


Fig 4. Transfer Symbols

2.2 Analysis of Standard Fault Trees

Analysis of standard fault trees is usually performed on two levels: a qualitative level and a quantitative level. After the creation of a SFT, minimal cut sets (MCSs) are usually obtained by performing qualitative analysis. Each MCS could contain a single event or multiple events combined by logic gates. The order of a minimal cut set defines the number of basic events that contribute to that minimal cut set. A 1st order MCS consists of a single basic event, i.e., a single failure event alone can cause the system failure. Therefore, this single component becomes a candidate for upgrade or to replicate. On the other hand, a 4th order MCS contains four basic events. The lower the order of a MCS the higher the importance of that MCS is. There are many algorithms available to perform the qualitative analysis of fault trees. A comprehensive list of these algorithms is available in (Ruijters & Stoelinga, 2015). The detail descriptions of these algorithms are out of scope of this paper. However, in this paper I briefly describe a popular algorithm — MOCUS (Fussel & Vesely, 1972).

MOCUS — Method of Obtaining Cut Sets

MOCUS (Fussel & Vesely, 1972) is a top-down approach and it is one of the primary standard fault tree analysis algorithms. Many other algorithms are developed based on this algorithm. This algorithm starts its operation with the top event of the fault tree and recursively explores the fault tree by expanding the intermediate events into their contributing basic events. This process continues until all the intermediate events are expanded and no more basic events are left in the tree.

Following are the steps of the algorithm (Walker, 2009):

1. Create a table where each row of the table represents a cut set and each column represents a basic event in the cut set.
2. Insert the top event of the fault tree in the first column of the first row.
3. Scan through the table, and for each fault tree gate:
    a. If the gate in an AND gate, then insert each of its input in a new column.
    b. If the gate is an OR gate, then insert each of its input in a new row.
4. Repeat step 3 until all the gates in the fault tree is explored and the table only contains the basic events.
5. Use Boolean laws to remove all redundancies within the table.

After performing all the above steps, each row of the resulting table will contain a minimal cut set. In terms of accuracy and speed of execution, MOCUS performs very well for smaller trees. However, the table size could grow much larger for large fault trees, hence this approach faces difficulties to analyze large fault trees. To illustrate the use of MOCUS algorithm for the analysis of fault trees, consider the fault tree of Fig. 5 used by Walker (2009).

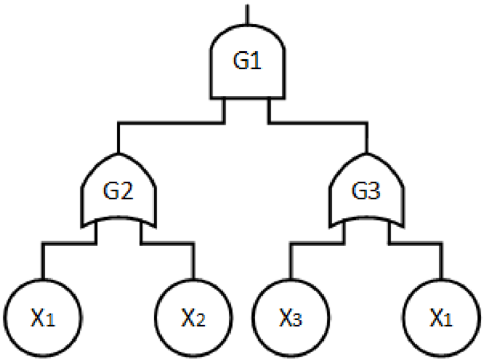


Fig 5. Fault tree to illustrate MOCUS algorithm

In the first step, a table is created and in the second step, the top event, G1 is put into the table.

G1

In the third step, this top event is expanded. As G1 is an AND gate, its inputs (G2 and G3) are put in a new column.

G2	G3

After that, G2 and G3 are expanded as well. As they are OR gates, their inputs are inserted in a new row.
G2 is expanded first:

X_1	G3
X_2	G3

G3 is expanded next:

X_1	X_3
X_1	X_1
X_2	X_3
X_2	X_1

Now no gates in the fault tree are left to be expanded. Boolean laws can be used to check if there exists any redundancies. Using the Idempotent law, two identical events in the same row can be reduced to just one, i.e. X_1 AND $X_1 \Leftrightarrow X_1$, thus:

X_1	X_3
X_1	
X_2	X_3
X_2	X_1

Using the Absorption law, a row can be eliminated if it contains all the elements of another row, so:

X_1	
X_2	X_3

This gives two minimal cut sets, one containing just X_1 and the other containing X_2 AND X_3.

Despite its limitations, MOCUS is one of the simplest and the most popular of FTA techniques, as evidenced by its 43 year life-span. As it is accurate and easy to understand, it is a good approach to analyse smaller fault trees; moreover, it makes an excellent foundation for further expansion – or extension – with new techniques. However, it is not the most efficient technique and algorithms like MICSUP (Pande, Spector, & Chatterjee, 1975), ELRAFT (Semanderes, 1971) tend to be faster.

Quantitative analysis of a fault tree can estimate the top event occurrence probability from the given failure rates/probabilities of basic failure events of the system (Vesely et al., 1981). In the quantification process, the basic events are usually assumed to be statistically independent. However, methodologies like Bobbio, Portinale, Minichino, & Ciancamerla (2001) can quantify fault trees with statistically dependent events. Usually, in FTA, as the top event is represented as the disjoint sum of the MCS, an approximate value of the probability of the top event can be determined by calculating the probability of each MCS and then adding them together, given that the probability of MCSs are small. In the Fault Tree Handbook (Vesely et al., 2002), this approximation is termed as “rare event approximation” and it is also stated that if the basic events probabilities are below 0.1 then this approximation are typically sufficiently accurate. In addition to this approximation, depending on the applications, different kinds of probabilities like time-dependent probabilities could be calculated provided that the proper failure distributions of the components/events are available.

To be able to perform quantitative analysis to get top event probability, the basic events are usually given one of the following types of data (Vesely et al., 2002):

1. an event occurrence probability or a component failure probability in some time interval,
2. unavailability of a component, and
3. a pure event probability.

A detail description of the quantitative analysis of the fault trees is available in (Ruijters & Stoelinga, 2015). Although the primary focus of the quantitative analysis of a fault tree is to estimate the top event probability, it is possible to estimate the probability of any intermediate events as well as the basic events. Dominance of the minimal cut sets could be determined based on the significance of their contribution to the top event. The cut set which contributes the most to the top event is considered as the most dominant. In addition to determining dominant MCS, importance of basic events could also be obtained in the similar way.

2.3 Limitations of Standard Fault Tree

Modern large and complex systems are becoming increasingly dynamic in nature. Such systems often have the capability to response to failure by partial self-repair. For example, they may equipped with backup components and can use the backup components when the primary component fails or they may continue their operation with degraded functionality or they can automatically detect and correct some specific types of errors. That means a system can operate in multiple modes, e.g. an aircraft can operate in take-off, flight, and landing phases. As a result, behaviour of a system (both nominal and potential failure behaviour) can change according to what state or mode of operation the system is. These behavioural uncertainties are increasingly prevalent as the complexity and scale of the systems increases, and dynamic behaviour can be observed in almost any large industrial systems. Dynamic behaviour of systems lead to different dynamic failure characteristics such as functional dependent events and priorities of failure events. Classical combinatorial fault trees are unable to model such dynamic scenarios.

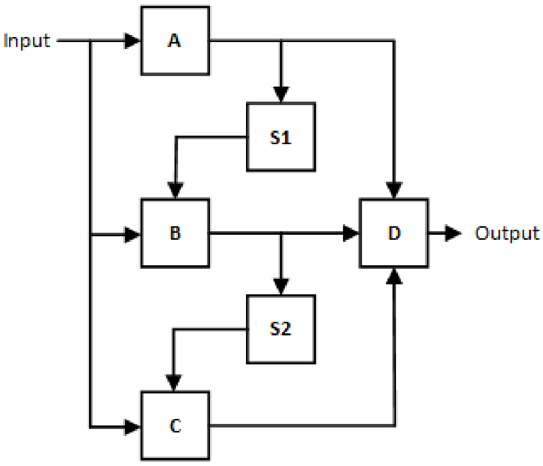


Fig 6. A triple-module redundant system

As a simple example of a system with dynamic behaviour, consider the generic standby system in Fig. 6. In this system, the components A, B and C could be any device. The primary component of the system (A) takes input from outside of the system, and after processing the input, it feeds the outcome to the system output (D). Component S1 is a monitoring sensor which monitors the activity of component A and activates the first standby component B upon detecting any output deviation from component A. Similarly, component S2 is another monitoring sensor which monitors the activity of component B and activates the second standby component C upon detecting any output deviation from component B.

Sensibly, failure of at most two components out of the three components can be tolerated by the system; however, failure of all three components can lead the system to failure. Performing a classical FTA on this system confirms that the system can only fail if any of the following occurs:

1. No input available to the system.
2. A, B, and C fail.
3. A and S1 fail.
4. A, B, and S2 fail.
5. D fails.

Therefore, each of these five conditions or combinations of these conditions is sufficient to cause the system failure. Initially, the above conditions appear to be sufficient and correct to define the failure behaviour of the system. For example, if we consider the first two conditions, then we can conclude that without input the system cannot operate, and if all the three components fail then the system cannot operate as well. But the situations described by conditions 3 and 4 are not as straightforward as conditions 1 and 2. According to condition 3, the system will fail if both components A and sensor S1 fail. In the case of condition 3, we can consider two possible scenarios: “A fails before S1 fails” and “S1 fails before A fails”. Now the question is: do both the scenario results in system failure? In the first scenario, if A fails before S1 fails, then it will not lead to the system failure because S1 has already activated the first standby component B, and thus failure of S1 after activation of B has no effect on the system failure behaviour. But in the second scenario, as S1 fails before A, omission of output from A therefore remains undetected and standby component B is never activated, and thus this will lead to system failure. Likewise, in case 4, if both A and B fail before S2 has failed, then it will not lead to system failure because in the meantime second backup component (C) will have been activated by sensor S2. Like condition 3, if S2 has served its purpose then the subsequent failure of S2 has no effect on the rest of the system. Therefore, condition 3 and 4 are unnecessarily pessimistic because it proposes that a failure of a sensor will always lead to system failure, but this is not always true; the outcome depends on the sequence of the component failures.

Now let us consider the effect of the behaviour of the sensors on the system behaviour. The sensors S1 and S2 are working by detecting an omission of output from component A and B respectively. If component B fails before A, then B will never be activated therefore S2 would never detect an omission of output from B, as a result C remains unused. This implies that the second condition, failure of A, B, and C causing system failure is a dangerously optimistic assumption, since B failing before A is sufficient to cause system failure, irrespective of the status of component C.

From the above example, it is clear that standard FTA is not capable of capturing sequence-dependent behaviour, and thus produces either unnecessarily pessimistic or excessively optimistic results. To overcome this limitation, a number of extensions to static fault trees such as dynamic fault trees (DFTs) (Dugan et al., 1992), State/Event Fault Trees (Kaiser et al., 2007), Temporal Fault Trees (Palshikar, 2002) and Pandora Temporal Fault Trees (Walker, 2009) have been proposed. Dynamic extensions of the classical fault trees along with other extensions are discussed in section 3.

Another limitation of SFT is that its quantitative analysis is performed based on fixed values of failure data of system components, and hence take it as guaranteed that the precise failure data of components are always available. However, in the very early stages of design, sometimes it is necessary to consider failure data of new components which have no available failure data. In some cases the exact choice of component has yet to be made and thus precise failure data could not possibly be known. As the outcomes of the quantitative analysis are entirely dependent on the accuracy of the data used in the analysis, the optimistic assumptions regarding the availability of precise failure data may produce misleading results or, in the worst case, the quantitative analysis may need to be discontinued due to the unavailability of the failure data. To overcome this limitation, fuzzy fault trees have been proposed by extending classical fault trees using fuzzy set theory. A brief description of fuzzy set theory based fault trees is provided in section 3.5.

Another issue with fault tree analysis is that it is primarily a manual process, i.e., performed manually either by a single person or a group of persons to produce some comprehensive documents to satisfy the safety requirements and to determine strategies to alleviate the effects of failures (Leveson, 1995). Although FTA can produce a significant amount of valuable knowledge about the system safety and reliability, due to its manual nature it has some limitations. Firstly, in the manual process the analyses are performed based on the informal system models. As the system design evolves, these informal models could rapidly become outdated. This presents challenges in maintaining the consistency and completeness of the assessment process. Secondly, as the system grows in size, the manual nature of the analysis process increases the risk of introducing error or producing incomplete results. Moreover, the manual analyses are time consuming and expensive, therefore the analyses are rarely carried out more than once even though the iterative process could produce more valuable information. Finally, the informal nature of these analyses does not allow a high degree of reusability of information, i.e., if a new analysis is required on the previously designed system or the system design is changed a bit, then the analysis is typically required to be started from the beginning meaning that it is difficult to reuse materials from a previous analysis.

To overcome the above mentioned limitation, a new field of model-based dependability assessment (MBDA) has emerged (Joshi et al., 2006; Walker et al., 2008). MBDA has attracted significant interest in the industry and academia over the last twenty years. In MBDA, the analysts perform their analyses on the design model of the system, which is created as part of a model-based design process. As the analyses are performed on a more formal model rather than a separate safety analyses model, it opens the avenue to automate part of the safety analysis process, e.g., automatically generating fault trees. In MBDA, as the analyses are performed on formal models, the analyses can be performed iteratively, which helps to generate more results and new results can be generated if the system design changes. This process is less time consuming and less expensive compared to manual approaches and due to its more structured nature, the risks of introducing errors in the analysis or producing incomplete results are reduced. Moreover, the MBDA techniques provide a higher degree of reusability by allowing parts of an existing system model, or libraries of previously analysed components, to be reused. More details of the MBDA techniques are described in section 4.

3. Extensions of Standard Fault Trees

3.1 Component Fault Trees

Classical static fault trees show the hierarchy of faults rather than the architectural hierarchy of the system components, therefore it is difficult to map fault tree elements to the corresponding system components. Component Fault Tree (CFT) is a modular version of the fault tree approach that extends classical fault trees by using real components in the tree structure. In the CFT approach, smaller fault trees for each system component are defined and those component fault trees are organised in a hierarchical structure according to the architectural hierarchy of the system (Kaiser, Liggesmeyer, & Mäckel, 2003). This enables closer synchronization of CFTs with the system model.

Like SFTs, CFTs use Boolean gates such as AND and OR gates. Moreover, CFTs utilise input output failure ports and internal failure events. Input and output failure ports are used to specify possible points of failure propagation and internal events are similar to that of the basic events of SFTs. CFTs differ from SFTs in that they allow multiple top events to be defined and represent repeating events only once. As a result, CFTs look more like a directed acyclic graph (Cause Effect Graph) than a tree like structure. However, standard fault tree algorithms could still be used to analyse CFTs both qualitatively and quantitatively as CFTs are still logical connections between faults and their causes. An example of a CFT is shown in Fig. 7.

The main advantage of using CFTs over SFTs is its hierarchical decomposition of systems to manage the complexity of modern systems. Classical FTA cannot benefit from the decomposition facilities because while creating a SFT, the analyst has to consider all levels of the system at once. In contrast, in CFT, smaller fault trees for each of the components are created and arranged in a hierarchical order according to the system architecture. As a result, different parts of the system can be developed and stored separately as part of the component definition in a library which facilitates a greater degree of reusability.

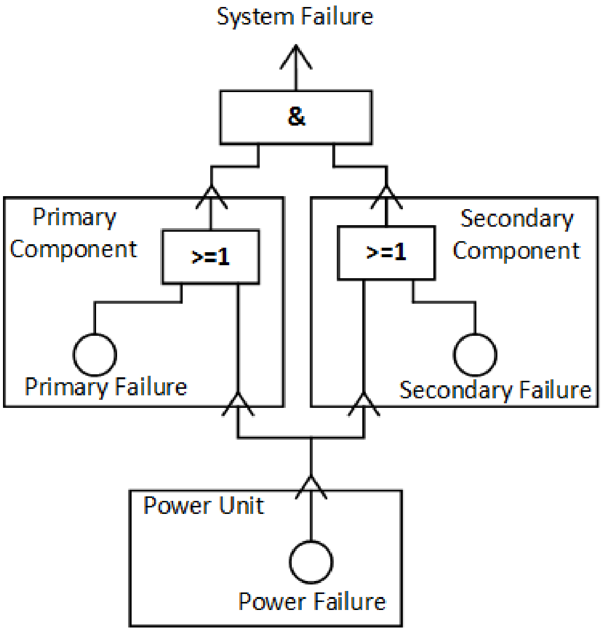
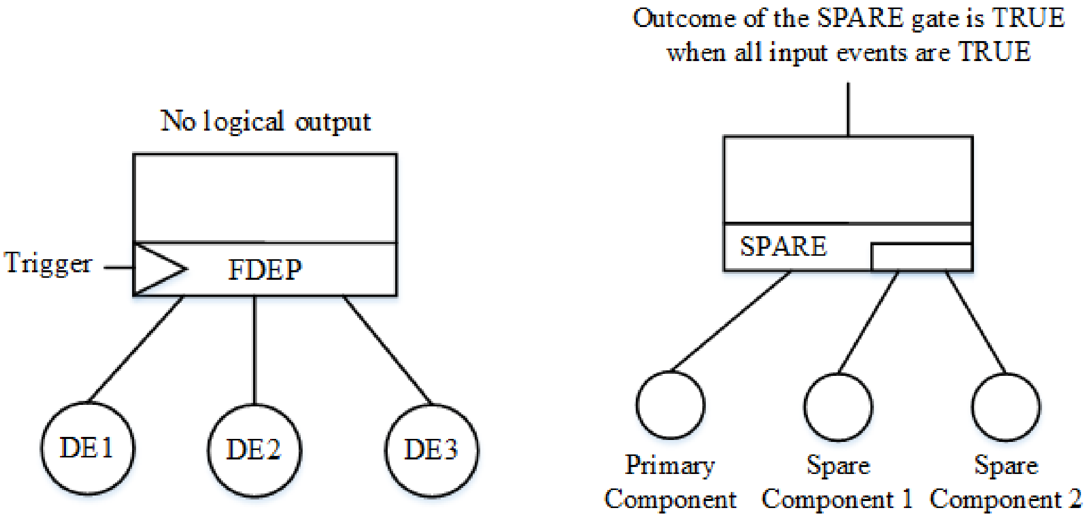


Fig 7. Component Fault Tree (Kaiser et al., 2003)

To be able to use CFTs in the architectural level of the systems, Grunske & Kaiser (2005b) have presented an approach to construct a system level CFT hierarchically based on a system architecture where all the components of the architecture are annotated with low-level CFTs. In order to analyse SaveCCM models (Åkerholm et al., 2007; Carlson, Håkansson, & Pettersson, 2006), Grunske (2006) have extended the procedure of creating hierarchical CFTs. A windows-based tool, ESSaReL (ESSaRel, 2005) is available to perform minimal cut set analysis and probabilistic evaluation of CFTs. Recently another tool called ViSSaAn (Visual Support for Safety Analysis) (Yang, Zeckzer, Liggesmeyer, & Hagen, 2011) has been developed based on a matrix view to allow improved visualisation of CFTs and efficiently representing information related to minimal cut sets of CFTs. Adler et al. (2011) have developed a meta-model to extract reusable CFTs from the functional architecture of systems specified in UML.

3.2 Dynamic Fault Trees

Dynamic Fault Trees (DFTs) (Dugan et al., 1992) are the most prominent dynamic extension of SFTs that enable a fault tree to capture sequence dependent dynamic behaviour. Two special gates, namely Functional Dependency (FDEP) gate and SPARE gate, are introduced as part of the Dynamic Fault Tree (DFT) to represent temporal behaviour of the systems.



(a) Functional Dependency (FDEP) gate (b) SPARE gate

Fig 8. Dynamic fault tree gates

The FDEP gate helps to design a scenario when the operations of some components of a system are dependent on the operation of another component of the system. For example, when many components of a system receive power from a single source of supply, then failure of the power supply would cause all the dependent components to fail. In the FDEP gate there is only one trigger event (either a basic event or an intermediate event) but there could be multiple functionally dependent events (see Fig. 8 (a)). The occurrence of the trigger event would force the dependent events to occur; by contrast, the occurrence of a dependent event would affect neither the trigger event nor the other dependent events. The FDEP gate is particularly useful for modelling networked systems, where communication between connected components takes place through a common network element, and failure of the common element isolates other connected components. This type of gate can also model interdependencies, which would otherwise introduce loops in the fault trees.

A SPARE gate is shown in Fig. 8(b). All the inputs to the SPARE gate are basic events, one of them acts as a primary component (left most input) and the others are the spare components. This gate designs a scenario where the spare components are activated in a sequence, i.e., if there are two spare components then the first spare will be activated in case the primary fails; if the first spare fails then the second one will be activated. The outcome of the SPARE gate becomes true if all the input events are true. The SPARE gate could model three types of spares: cold spares, warm spares, and hot spares. The failure rate of each of the spare components is affected by the mode they are in and this effect is modelled by a dormancy factor. In the cold spare mode the spare components are deactivated until they are required therefore the dormancy is closer to zero. In contrast, in the hot spare mode, spare components are always active but serve their functionality when the primary fails; as a result the failure rate of a spare component is the same as an active component even if it is not in service, and therefore a spare component has a dormancy factor close to one. In warm spare mode, the spare components are neither on nor off, instead they are kept in-between these two states, i.e., components are kept in a reduced readiness state until required. The dormancy factor of a component in warm spare mode is considered somewhere in-between the dormancy factor of cold and hot spare modes (e.g., 0.5). Multiple SPARE gates can share a pool of spare components. In this case, if the primary component of any of the SPARE gates fails, it is then replaced by the first available spare component (i.e., neither failed nor already occupied by another SPARE gate).

DFTs also use two other gates to model sequences of events: the Priority-AND (PAND) gate, which is true only if its input events occur in a particular sequence (typically left to right), and the Sequence-Enforcing gate (SEQ), which imposes a sequence on its events such that they must occur in that order. This latter gate can be viewed as a type of cold SPARE gate and so is not often used.

DFTs are intended to perform quantitative reliability analysis of dynamic systems, and consequently they have limited support for qualitative analysis. The lists of approaches to perform qualitative and quantitative analysis of DFTs are available in (Ruijters & Stoelinga, 2015). For probabilistic evaluation, DFTs are typically transformed into equivalent Markov chains and quantified based on exponential distribution of failure behaviour of components (Boudali, Crouzen, & Stoelinga, 2007a, 2007b, 2010; Dugan, Bavuso, & Boyd, 1993). Alternative solutions to DFTs have also been proposed, e.g. an algebraic framework (Merle, Roussel, & Lesage, 2011, 2014; Merle, Roussel, Lesage, & Bobbio, 2010) to model the dynamic gates of DFTs. Moreover, Monte Carlo simulation based methods (Boudali, Nijmeijer, & Stoelinga, 2009; Ejlali & Miremadi, 2004; Manno, Chiacchio, Compagno, D'Urso, & Trapani, 2012; Rao et al., 2009; Zhang & Chan, 2012), Petri Nets based approaches (Codetta-Raiteri, 2005; Zhang, Miao, Fan, & Wang, 2009), Bayesian Networks based approaches (Boudali & Dugan, 2005; Boudali & Dugan, 2006; Marquez, Neil, & Fenton, 2008; Montani, Portinale, Bobbio, & Codetta-Raiteri, 2008), and modularisation approaches (Anand & Somani, 1998; Chiacchio et al., 2013; Gulati & Dugan, 1997; Huang & Chang, 2007; Manian, Dugan, Coppit, & Sullivan, 1998; Pullum & Dugan, 1996; Yevkin, 2011) have also been developed to quantify DFTs.

3.3 Pandora Temporal Fault Trees

Pandora is an extension of classical fault trees, which makes conventional fault trees capable of dynamic analysis (Walker, Bottaci, & Papadopoulos, 2007; Walker, 2009). Pandora augments SFTs with new temporal gates and defines temporal laws to allow qualitative analysis, and thus overcome the limitations of FTA in expressing sequence-dependent behaviour. The basis of Pandora is the redefinition of the long-established Priority-AND (PAND) gate (Fussell, Aber, & Rahl, 1976).

Pandora assumes that the occurrence of the events are instantaneous, i.e., go from 'non-fail' to 'fail' with no delay, and once events occur, they cannot be repaired to go to a non-occurred state. Given this, there are three possible temporal relationships between two events X and Y:

- *before* – X occurs first, Y occurs second
- *after* – Y occurs first, X occurs second
- *simultaneous* – X and Y occur at the same time

To represent these three temporal relations between events, i.e., to capture the sequence between the occurrence of events, Pandora introduces three temporal gates: Priority-AND (PAND), Priority-OR (POR), and Simultaneous-AND (SAND). The PAND gate represents a sequence between events X and Y where event X must occur before event Y, but both the events must occur. The POR gate also represents a sequence between the events, but it specifies an ordered disjunction rather than an ordered conjunction, i.e., event X must occur before event Y if event Y occurs at all. The SAND gate models the simultaneous occurrence of events.

Other than the temporal gates, Pandora also uses the Boolean gates of the SFTs. In order to relate the Boolean gates with the temporal gates, Pandora defines a set of new temporal laws (Walker, 2009). These laws enable Pandora to perform a qualitative analysis of temporal fault trees (TFTs). By performing qualitative analysis, Pandora can obtain the minimal cut sequences (MCSQs), which are the smallest sequences of basic events necessary and sufficient to cause the top event. Using Pandora, the causes of failure of the triple-module redundant system of Fig. 6 would be as follows:

1. No input available to the system.
2. A fails before B fails and C fails.
3. S1 fails before A fails.
4. S2 fails before B or A fails.
5. Dormant failure of B before A fails.
6. D fails.

These minimal cut sequences are more meaningful and refined than the cut sets obtained by the SFT in section 2.3. Particularly, the effects of ordering of events can now be taken into account.

The major advantages of Pandora are that by performing qualitative analysis it can provide useful information about system failure behaviour in the absence of precise quantitative data. Quantitative analysis of Pandora fault trees is possible, given that quantitative data about component failure behaviour is available. The analytical solution (Edifor, Walker, & Gordon, 2012, 2013) to Pandora defines mathematical expressions for probabilistic evaluation of Pandora TFTs. This approach considers that the system components have exponentially distributed failure data. Monte Carlo simulation based method (Edifor, Walker, Gordon, & Papadopoulos, 2014) has also been developed to quantify the Pandora TFTs. Petri Nets have also been used to probabilistically evaluate Pandora TFTs (Kabir, Walker, & Papadopoulos, 2015). In the Petri Net based approach, the Pandora TFTs are transformed into Generalised Stochastic Petri Nets (GSPNs) (Marsan, Balbo, Conte, Donatelli, & Franceschinis, 1996). GSPN models are subsequently simulated to obtain the probability of the top event. A Bayesian Network based method (Kabir, Walker, & Papadopoulos, 2014) has also been proposed for probabilistic evaluation of Pandora TFTs. This approach can work with any kind of distributions of data. All the existing approaches can only be used to perform predictive analysis of systems but the Bayesian Network based approach can also be used for diagnostic analysis of systems in addition to predictive analysis. Diagnostic analysis involves calculating and updating the posterior probability of basic events given observed evidence of the system failure.

3.4 State Event Fault Trees

As already mentioned, the classical combinatorial FTA is suitable for modelling static behaviour of systems but is not suitable for modelling dynamic behaviour. To overcome this limitation, State-Event Fault Trees (SEFT) extend conventional FTA by adding capabilities for representing states and events to fault trees, thus overcoming the limitation of standard FTA by using state-based system models in the analysis (Grunske, Kaiser, & Papadopoulos, 2005). SEFTs can also be seen as an extension of CFTs with probabilistic finite state models. Therefore, elements/components in the system architecture are modelled with a set of states and probabilistic transitions between these states (see Fig. 9). A state is graphically represented as a rounded rectangle and considered as a condition that lasts over a period of time, whereas an event is graphically represented as a solid bar and considered as an instantaneous phenomenon that can cause a state transition.

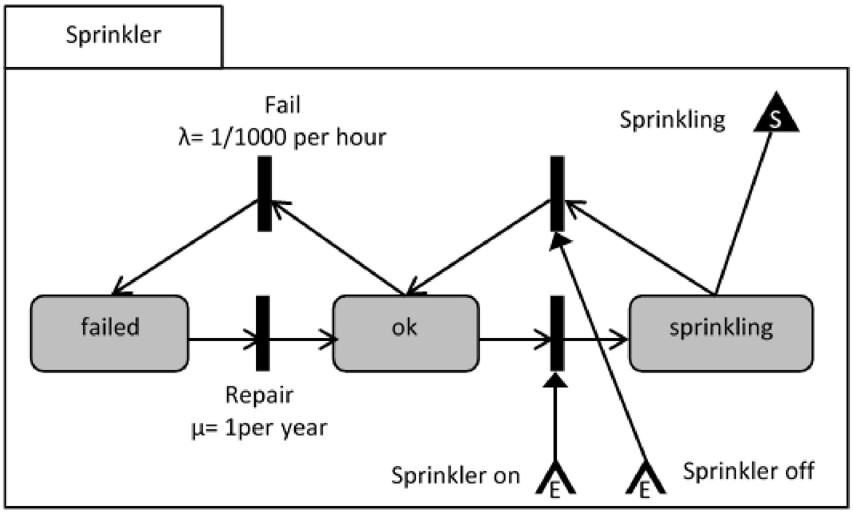


Fig 9. State Event Fault Tree

Grunske, Kaiser, & Papadopoulos (2005) have described a method to construct a SEFT from a system architecture where each of its elements has their SEFT defined. This method identifies the relationship between components based on name-matching of the state and event ports as well as the data and control flow specified in the architecture.

As SEFTs make a distinction between causal and sequential transitions they therefore provide separate types of ports for them. Sequential transition applies to states which specify predecessor / successor relation between states; in contrast, causal transition applies to events which define a causal (trigger/guard) relationship between events. As events are explicitly represented in SEFTs, it is possible for one or more combinations of events to cause another event. To combine the events Boolean gates (e.g., AND and OR) of standard fault trees could be used. SEFTs can also include temporal information in the failure expression like assigning events with deterministic or probabilistic delays by using

Delay Gates. History-AND, Priority-AND and Duration gates are employed to specify causal and sequential relations between events with respect to time. For example, the History-AND gate checks whether an event has occurred in the past, the Priority-AND gate can check in what order events have occurred, and the Duration gate can ensure that the system has been in a certain state for a specified amount of time.

Like CFTs, SEFTs can also be structured as a hierarchy of architectural components, where ports are used to specify the interactions between the components. SEFTs follow the same procedure as that of FTAs during modelling of system failure, i.e., the analysis starts with a system failure and works backwards to determine its root causes. As it uses state-based behaviour descriptions it can use pre-existing state-based models from the system design, which results in a greater degree of reusability compared to standard fault trees. However, it is no longer possible to analyse SEFTs using traditional fault tree algorithms because of the state-based representation they have used. Different types of techniques are now required to convert SEFTs into other representation like Petri Nets or Markov chains for quantitative evaluation of SEFTs. Steiner, Keller, & Liggesmeyer (2012) have proposed a methodology to create and analyse SEFTs using the ESSaRel tool. Consequently, the SEFT models are converted to Deterministic Stochastic Petri Nets (DSPNs) (Marsan & Chiola, 1987), then the analysis of the DSPN models can be performed using separate DSPN analyser like TimeNET (German & Mitzlaff, 1995). In the conversion process the whole system is required to be considered, i.e., all the components and subcomponents with their own state-based behaviour are to be considered, which would lead to a combinatorial state-space explosion. This problem can be remedied to some extent by using a combinatorial-FTA like algorithm for the static part of the system and using more efficient algorithms for the dynamic part of the system. For example, to avoid state space explosion, a modularised technique has been proposed by Förster & Kaiser (2006), which identified independent dynamic modules from other static modules of a system, and subsequently used Binary Decision Diagrams and DSPN for the analysis of static and dynamic modules, respectively. However, the performance of the dual-analysis technique will depend on the type and complexity of the system being analysed. Although all the existing methodologies for the analysis of SEFTs are for the quantitative analysis, recently, a method has been proposed by Roth & Liggesmeyer (2013) for qualitative analysis of SEFTs. In addition, formal semantics for SEFTs based on guarded interface automata and a method to obtain minimal cut sequence based on these semantics have been proposed by Xu, Huang, Hu, Wei, & Zhou (2013).

3.5 Fuzzy Fault Trees

During quantitative analysis, in standard FTA, failure rates or failure probabilities of system components are usually considered to be constant. But for many large and complex systems, it is often very difficult to obtain precise failure data due to lack of knowledge, scarcity of statistical data, ambiguous component behaviour, and operating environment of the system (Liang & Wang, 1993; Singer, 1990). Fuzzy set theory has been proven effective in solving problems where precise data are not available and in making decisions from vague information (Suresh, Babar, & Raj, 1996; Zadeh, 1965). To address the problem of uncertain failure data, fuzzy set theory was firstly used in FTA by Tanaka, Fan, Lai, & Toguchi (1983), where failure probabilities of the basic events of the fault tree were represented as trapezoidal fuzzy numbers and the fuzzy extension principle was used to estimate the probability of the top event.

Fuzzy set theory has been used in fault tree analysis in different ways. A literature survey on different variants of fault trees with fuzzy numbers could be found in (Mahmood, Ahmadi, Verma, Srividya, & Kumar, 2013; Ruijters & Stoelinga, 2015). The main idea behind fuzzy fault tree analysis is to use a fuzzy representation of the failure data instead of crisp values and then evaluate the top event as a range of possible values. In this process, opinions about components failure probability are usually obtained from experts. Due to the complexity of the systems and the vagueness of the events, the experts cannot provide the exact numerical values regarding the failure probability of components; instead they give their opinion in linguistic terms. The values of linguistic variables are words or sentences in natural languages. Linguistic variables play a vital role in dealing with complex or vague situations. Once an expert provides his/her opinion about the failure probability of an event in linguistic terms, then this must be mapped to corresponding quantitative data in the form of a membership function of fuzzy numbers. For example, Fig. 10 shows membership functions of the linguistic variables in the triangular form.

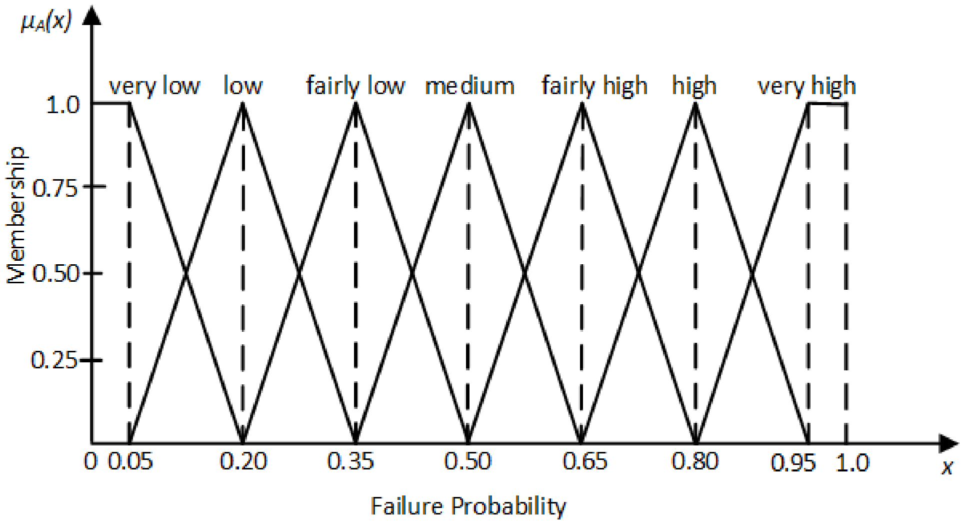


Fig 10. Example of fuzzy membership functions for a set of linguistic variables (Kabir, 2016)

After defining fuzzy failure rates or failure probabilities of all basic events, fuzzy operators for the fault tree's gates are defined and subsequently, all the MCSs are quantified using the fuzzy failure rates of basic events. These values are used to obtain the top event probability. As the fuzzy numbers are used in the quantification process, the top event probability is obtained as fuzzy numbers.

After the introduction of the fuzzy fault tree by Tanaka et al. (1983), further extensive research on fuzzy fault tree analysis was performed by Misra & Weber (1990) and Liang & Wang (1993) based on this. At the same time, Gmytrasiewicz, Hassberger, & Lee (1990) and Singer (1990) have also analysed fault trees based on fuzzy set theory. Fuzzy set theories and the expert elicitation have been combined by Lin & Wang (1997) to evaluate the reliability of a robot drilling system. Fuzzy set theory based FTA (FFTA) has been used to analyse the reliability of a variety of systems, for example, Yuhua & Datao (2005) have used FFTA to evaluate the failure probability of oil and gas transmission system. An intuitionistic fuzzy sets based method has been used in (Shu, Cheng, & Chang, 2006) for the failure analysis of the printed circuit board assembly. Ferdous, Khan, Veitch, & Amyotte (2009) have proposed a computer-aided fuzzy fault tree analysis method. Tyagi, Pandey, & Kumar (2011) have applied FFTA in reliability analysis of an electric power transformer. Recently, FFTA has been used to evaluate the probability of the fire and explosion in crude oil tanks (Wang, Zhang, & Chen, 2013) and Rajakarunakaran, Kumar, & Prabhu (2015) have applied FFTA for risk evaluation of an LPG refuelling station. In addition to classical fault trees, fuzzy numbers have also been used in dynamic fault trees (e.g. (Li, Huang, Liu, Xiao, & Li, 2012; Li, Mi, Liu, Yang, & Huang, 2015; Verma, Srividya, Prabhudeva, & Vinod, 2006; Yang, 2011)) and temporal fault trees (e.g. (Kabir, Edifor, Walker, & Gordon, 2014; Kabir, Walker, Papadopoulos, Rüde, & Securius, 2016)).

In addition to the extensions of the fault trees described in the above subsections, there are different other extensions such as repairable fault trees (Balakrishnan & Trivedi, 1995; Beccuti, Codetta-Raiteri, Franceschinis, & Haddad, 2008; Codetta-Raiteri, Franceschinis, Iacono, & Vittorini, 2004), temporal fault trees (Gluchowski, 2007; Palshikar, 2002; Wijayarathna & Maekawa, 2000), Stochastic Hybrid Fault Tree Automaton (SHyFTA) (Chiacchio et al., 2016), and non-coherent fault trees (Beeson, 2002; Contini, Cojazzi, & Renda, 2008) etc. Interested readers are referred to (Ruijters & Stoelinga, 2015) for more information about these fault tree extensions.

Table 1. Tools Support for Different Fault Tree Extensions

Approach	Tool Support
SFT	Isograph FaultTree+ (Isograph, 2016); OpenFTA (Auvation, 2016); SAPHIRE (Idaho National Laboratory, 2016); ALD RAM Commander (ALD, 2014); ReliaSoft BlockSim (ReliaSoft, 2016); TDC FTA (TDC Software, 2016); EPRI CAFTA (EPRI, 2013); LOGAN FTA (LOGAN, 2016); ELMAS (Ramentor, 2016); PTC Windchill FTA (PTC, 2016); ITEM ToolKit (ITEM Software, 2016); CARE FTA (BQR, 2015); GRIF-Workshop (GRIF, 2016); RiskSpectrum FTA (RiskSpectrum, 2015)
DFT	Galileo (Dugan, Sullivan, & Coppit, 2000); Isograph FaultTree+ (Isograph, 2016); DFTCalc (Arnold, Belinfante, Van der Berg, Guck, & Stoelinga, 2013); MatCarloRe (Manno et al., 2012); DFTSim (Boudali et al., 2009); DIFtree (Dugan, Venkataraman, & Gulati, 1997); RAATSS (Chiacchio, 2012, Manno et al., 2014); SHyFTA (Chiacchio et al., 2016); RADYBAN (Montani et al., 2008); DRSIM (Rao et al., 2009); DFT2GSPN (Codetta-Raiteri, 2005)
CFT	ESSaRel tool (ESSaRel, 2005)
Pandora TFT	Pandora (Walker & Papadopoulos, 2009)
SEFT	ESSaRel tool (ESSaRel, 2005); SEFTAnalyzer (Roth & Liggesmeyer, 2013)
FFT	FuzzyFTA (Guimarees & Ebecken, 1999)

4. Model Based Dependability Analysis and Application of FTA in MBDA

This section reviews different model based dependability analysis approaches which applied FTA as a means for their analysis technique and automatically or semi-automatically generates fault trees from extended system models.

4.1 Failure Propagation and Transformation Notation

Failure Propagation and Transformation Notation (FPTN), which overcomes many of the limitations of FTA, is the first modular and graphical method to specify failure behaviour of systems with complex architectures. FPTN was created to provide a simple and clean notation to reflect the way in which failures within the system interact along with the system architecture (Fenelon & McDermid, 1993). The basic unit of the FPTN is a “module” and usually is represented by a simple box with a set of input (incoming) and output (outgoing) failure modes. These inputs and outputs are connected to other FPTN modules and a module can contain a number of sub-modules to form a hierarchical structure. Failures can be propagated directly through one FPTN module to another or can be transformed from one type to another. These failures modes could be classified into a number of broad categories, like (Fenelon & McDermid, 1993):

- Timing Failure

  – too early (te), too late (tl) etc.

- Value failures (v)
- Commission (c)
- Omission (o)

Example of an FPTN module is shown in Fig. 11. Inside the box, representing a module, there are some standardised sections, each of which consists of some standard attributes like the header section consists of modules name, and its criticality etc. The most important part is the module specification section (second section) which describes the relationship between input and output failure modes by defining the failure propagations, transformations, generations and detections. This section is important because it defines how the module is affected by the other modules or environment and how other modules or environment are likely to be affected by this module. Each output failure mode of a FPTN module is specified by a single Boolean equation as a sum-of-products of input failure modes (like minimal cut sets of FTA). Therefore a FPTN module could be considered as a forest of trees. In the example FPTN module, one of the output failure modes, Sprinkling: o is specified as Sprinkling: o = Smoke_Detected: tl + Smoke_Detected: o; i.e., the omission of Sprinkler output could be caused if Smoke Detected input is too late (Smoke_Detected: tl) or omission of Smoke Detected input (Smoke_Detected: o) occurs.

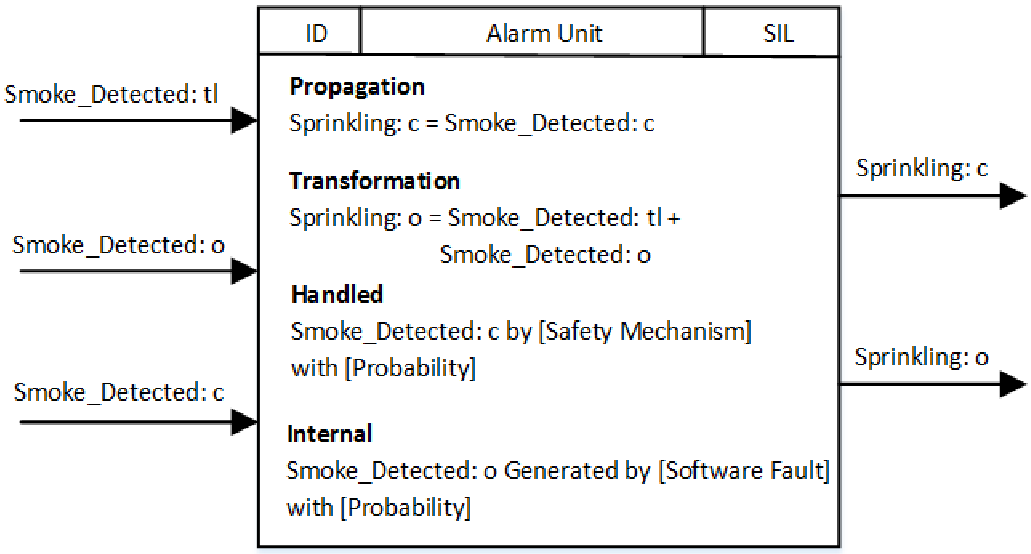

ID
Alarm Unit
SIL
Smoke_Detected: tl
Smoke_Detected: o
Smoke_Detected: c
Propagation
Sprinkling: c = Smoke_Detected: c
Transformation
Sprinkling: o = Smoke_Detected: tl + Smoke_Detected: o
Handled
Smoke_Detected: c by [Safety Mechanism] with [Probability]
Internal
Smoke_Detected: o Generated by [Software Fault] with [Probability]
Sprinkling: c
Sprinkling: o

Fig 11. Example FPTN Module (Grunske & Han, 2008)

Other than propagating and transforming failures modes from one form to another, an FPTN component can also generate failure modes (internal failure modes) or can handle some of the existing failure modes. These capabilities make FPTN usable as both an inductive (creating FMECA) and a deductive (creating FTA) analysis method. Usually, to evaluate FPTN modules, a fault tree is created for each of the outgoing failure modes. The process of using FPTN and fault trees to perform model-based safety analysis was shown in (Grunske & Kaiser, 2005b). To illustrate the idea, consider the steam boiler system shown in Fig. 12.

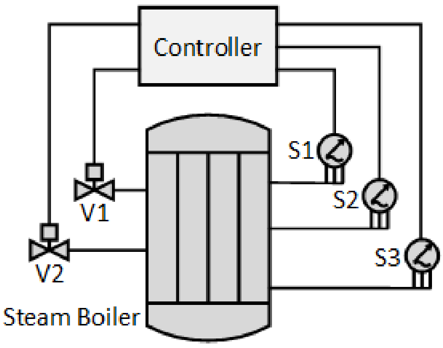


Fig 12. Steam Boiler System (Grunske & Kaiser, 2005b)

In the boiler system, there are one steam boiler, three pressure sensors (S1, S2, and S3), two valves (V1 and V2), and one controller. In normal operating condition, the sensors monitor the pressure on the steam boiler and report it back to the controller. The controller then implements a two-out-of-three voting mechanism for the pressure sensors and if pressure is higher than a certain threshold values then the controller sends command to open the valves to release the extra pressure.

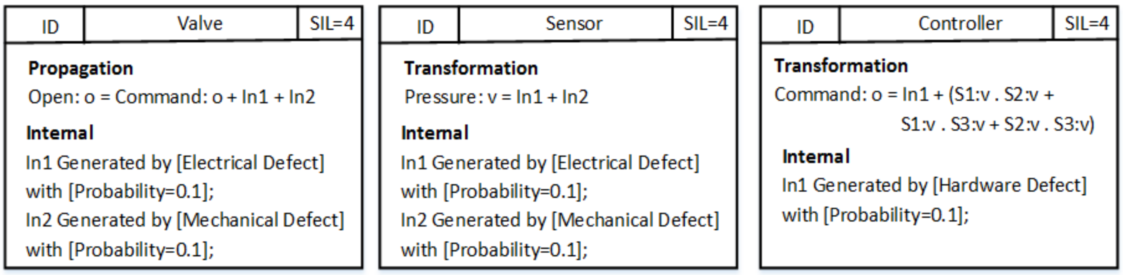

ID
Valve
SIL=4
Propagation
Open: o = Command: o + In1 + In2
Internal
In1 Generated by [Electrical Defect] with [Probability=0.1];
In2 Generated by [Mechanical Defect] with [Probability=0.1];
ID
Sensor
SIL=4
Transformation
Pressure: v = In1 + In2
Internal
In1 Generated by [Electrical Defect] with [Probability=0.1];
In2 Generated by [Mechanical Defect] with [Probability=0.1];
ID
Controller
SIL=4
Transformation
Command: o = In1 + (S1:v . S2:v + S1:v . S3:v + S2:v . S3:v)
Internal
In1 Generated by [Hardware Defect] with [Probability=0.1];

Fig 13. FPTN modules of the components of the system in Fig. 12 (Grunske & Kaiser, 2005b)

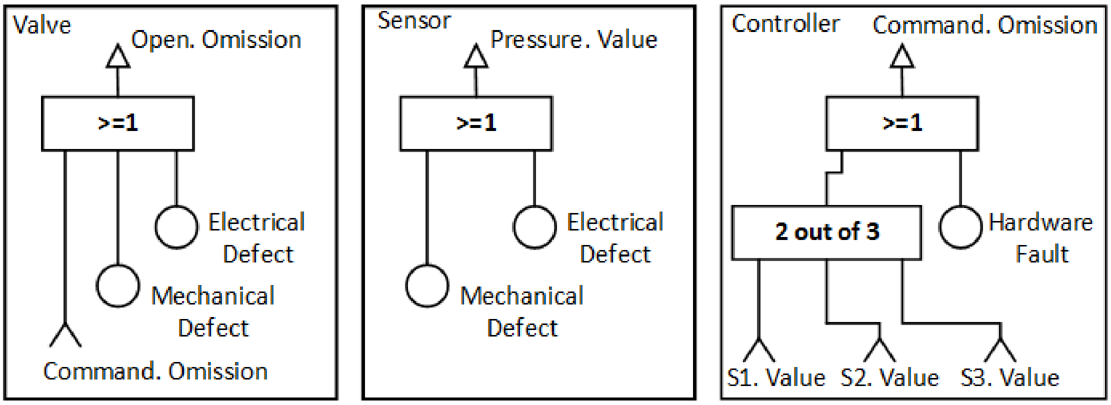


Fig 14. Component fault trees of the components of the steam boiler system (Grunske & Kaiser, 2005b)

Following the instructions presented in (Grunske & Kaiser, 2005b), at first, the FPTN modules for the components of the boiler system are created, and subsequently the CFTs are created from the FPTN-modules. Based on the system architecture and data flow inside the architecture, the CFTs of the lower level components are combined to obtain the CFT of the system as a whole, and shown in Fig. 15. For full description of the process of constructing system level CFTs from component level CFTs, the readers are referred to (Grunske, Kaiser, & Reussner, 2005; Grunske & Kaiser, 2005a, 2005b).

As FPTN modules are created alongside the design of the system, if the system model changes then the failure model also changes. Potential flaws and problems can be identified from the analysis of FPTN modules and these flaws could be rectified in the subsequent design iterations. Although FPTN provides systematic and modular notations for representing failure behaviour of systems, in its classical form it can only perform static analysis, not dynamic. However, recently, Niu, Tang, Lisagor, & McDermid (2011) have extended the classical FPTN notations with temporal information, thus making it capable of performing dynamic analysis. The basic structure of the FPTN module was kept as it is in the classical form, however, temporal logic, referred to as Failure Temporal Logic (FTL) is used instead of Boolean logic to specify equations to express the relationships between input and output failure modes to a FPTN module. As nothing is changed except the logic to represent equations, the new modelling framework is still suitable to be used in hierarchical structure, and now the outcome of the analysis would be temporal fault trees with Minimal Cut Sequences (MCSQs) instead of classical fault trees with minimal cut sets.

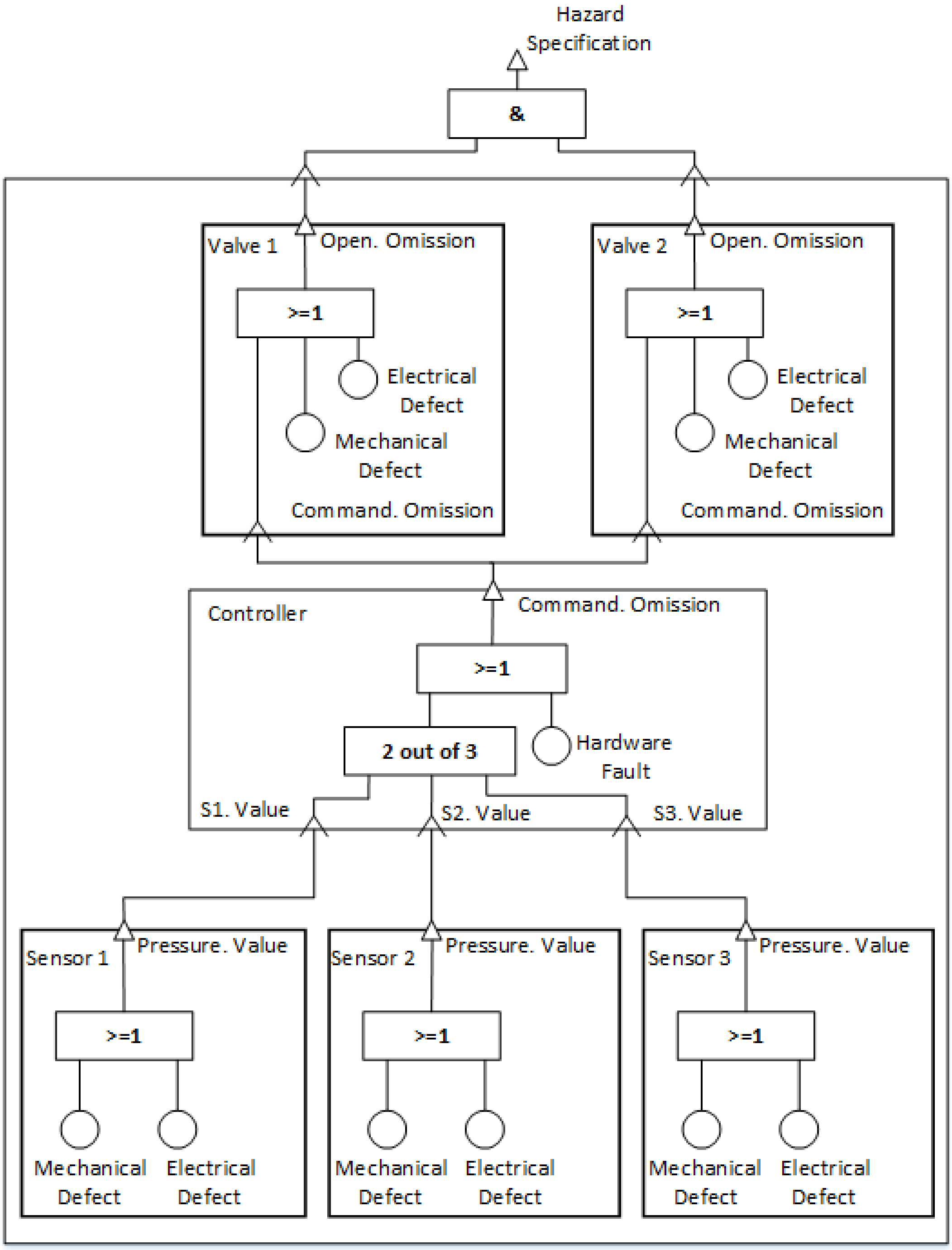
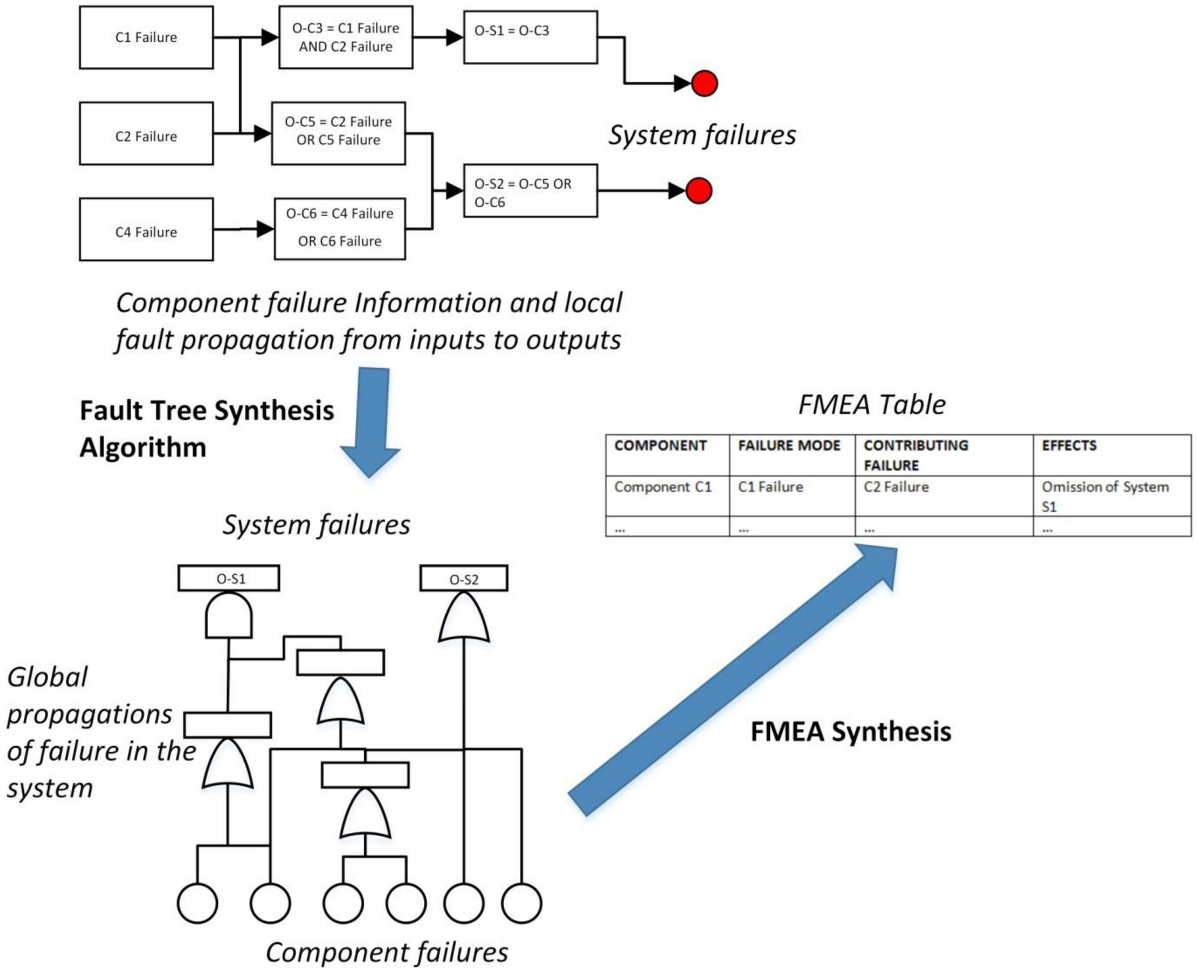


Fig 15. Component Fault Tree of the Steam Boiler System

4.2 Hierarchically Performed Hazard Origin and Propagation Studies

Hierarchically Performed Hazard Origin & Propagation Studies or HiP-HOPS (Papadopoulos & Mcdermid, 1999; Papadopoulos, 2000) is one of the more advanced and well supported compositional model based dependability analysis techniques. It can provide similar functionality to CFTs and FPTN but with more features and a greater degree of automation. Moreover, it can automatically generate fault trees, FMEA tables, perform quantitative analysis on fault trees, and has the ability to perform multi-objective optimisation of the system models (Papadopoulos et al., 2011, 2016). It can semi-automatically allocate safety requirements to the system components in the form of Safety Integrity Levels (SILs) which automates some of the processes for the ASIL allocation specified in ISO26262 (ISO, 2011).

HiP-HOPS can analyse any system that has identifiable components and some material, energy or dataflow transactions among the components. To analyse a system using the HiP-HOPS tool, analysts have to provide an annotated system model as an input to the tool. In order to model and annotate system components with dependability related information, HiP-HOPS uses popular modelling tools such as Matlab Simulink or Simulation X. The dependability related information includes component failure modes and expressions for output deviations, which describe how a component can fail and how it responds to failures occurred in other parts of the system. By using the annotated system model as an input, HiP-HOPS can automatically generate dependability analysis artefacts such as fault trees and FMEAs by tracing the component failure propagations through the system architecture. A generic overview of the HiP-HOPS dependability analysis technique is shown in Fig. 16.


C1 Failure
C2 Failure
C4 Failure
O-C3 = C1 Failure AND C2 Failure
O-C5 = C2 Failure OR C5 Failure
O-C6 = C4 Failure OR C6 Failure
O-S1 = O-C3
O-S2 = O-C5 OR O-C6
System failures
Component failure Information and local fault propagation from inputs to outputs
Fault Tree Synthesis Algorithm
FMEA Table
System failures
O-S1
O-S2
Global propagations of failure in the system
FMEA Synthesis
Component failures

COMPONENT	FAILURE MODE	CONTRIBUTING FAILURE	EFFECTS
Component C1	C1 Failure	C2 Failure	Omission of System S1
…	…	…	…

Fig 16. A generic overview of HiP-HOPS Technique (Sharvia, Kabir, Walker, & Papadopoulos, 2015)

HiP-HOPS analysis consists of three main phases:

1. System modelling and failure annotation
2. Fault Tree synthesis
3. Fault Tree analysis and FMEA synthesis

The system modelling and failure annotation phase allows analysts to provide information to the HiP-HOPS tool on how the different system components are interconnected and how they can fail. The architectural model of the system shows the interconnections between the components of system and the architecture can be arranged hierarchically, i.e., the system consists of different subsystems and subsystems have their own components. For instance, the architectural model of the boiler system of Fig. 12 is shown in Fig. 17.

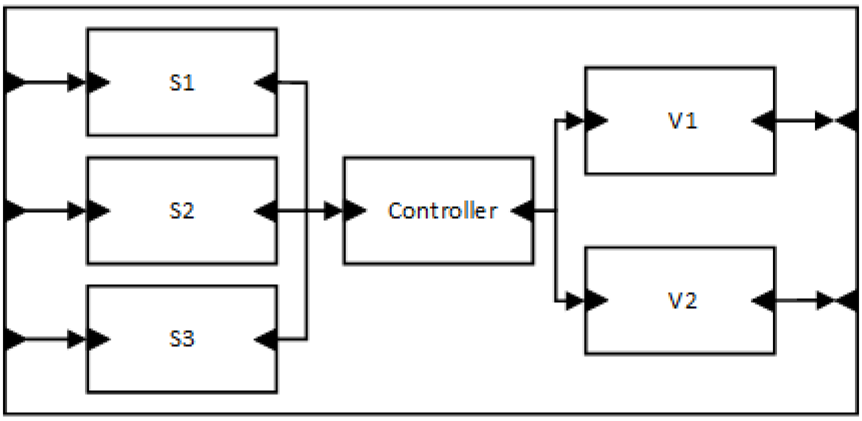


Fig 17. Architecture of the Boiler System of Fig. 12

HiP-HOPS considers that an output deviation of a component could be caused by an internal failure, an input failure, or combination of both. Therefore, local failure information for each of the components needs to be entered which describes what can go wrong with a component and how a component responds to a failure propagated from other parts of the system. The failure information of components are provided as a set of Boolean expressions. These expressions describe how the output deviations of a component can be caused either by internal failure of that component or by corresponding input deviations of the component. Such deviations could be omission of output, unexpected commission of output, incorrect output, or too late or early arrival of output. Other than the failure behaviour of the components in the form of logical expressions, quantitative data (failure rate or failure probability) can also be provided during annotating components. Numerical data are used in quantitative analysis. When all the logical and numerical information regarding the failure behaviour of the components are specified, then the component can be saved in a library to facilitate future reuse. Failure modes of the boiler system in Fig. 12 and their annotations are shown in Table 2 and 3 respectively.

Table 2. Internal failure modes of the components of the boiler system

Component	Failure Modes	Probability
V1	Electrical defect	0.1
	Mechanical Defect	0.1
V2	Electrical defect	0.1
	Mechanical Defect	0.1
Controller	Hardware Fault	0.1
S1	Electrical defect	0.1
	Mechanical Defect	0.1
S2	Electrical defect	0.1
	Mechanical Defect	0.1
S3	Electrical defect	0.1
	Mechanical Defect	0.1

Table 3. Failure mode expressions for the boiler system

Component	Output Deviations	Failure Expressions
V1	Omission.V1-Open	Electrical defect + Mechanical Defect + Omission-Controller. Command
V2	Omission.V2-Open	Electrical defect + Mechanical Defect + Omission-Controller. Command
Controller	Omission-Controller. Command	Hardware Fault + Omission.S1-Output. Omission.S2-Output + Omission.S2-Output. Omission.S3-Output+ Omission.S1-Output. Omission.S3-Output
S1	Omission.S1-Output	Electrical defect + Mechanical Defect
S2	Omission.S2-Output	Electrical defect + Mechanical Defect
S3	Omission.S3-Output	Electrical defect + Mechanical Defect

The synthesis phase starts its operation with a deviation of system output (top event) and traverses the system architecture backwards, i.e., from the system level to the component level, to examine the input components leading to the system output, and from there to the inputs of those components, and so forth. In this way the tool traverses the whole architecture and creates local fault trees for the interconnected components. The traversal continues until no connected components remains. After that, the tool goes back and combines the local fault trees into a single fault tree which represents all the possible combinations of component failure that can lead to the system failure. Fault tree of the failure behaviour of the boiler system produced by the HiP-HOPS tool is shown in Fig. 18.

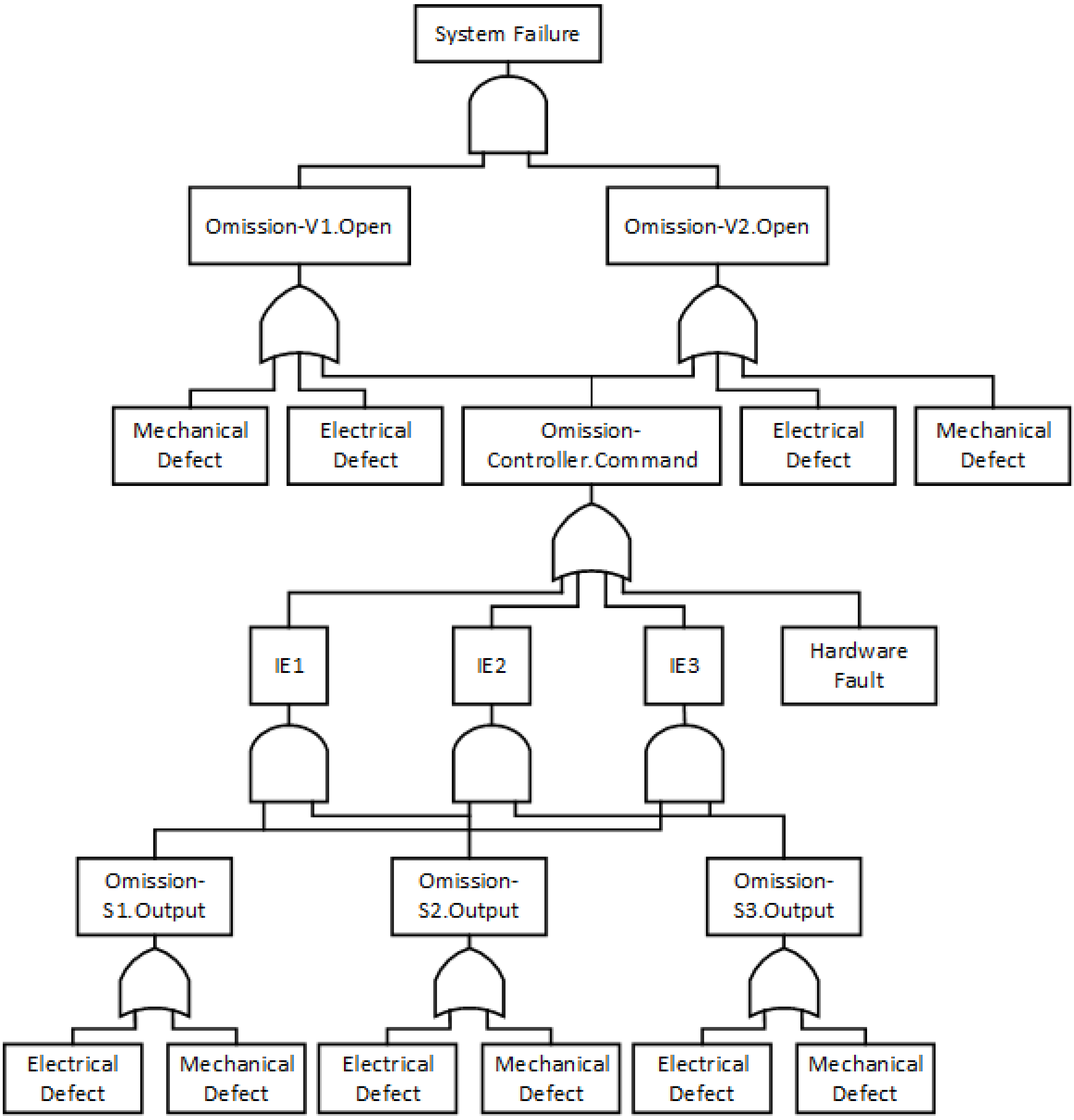


Fig 18. Fault tree of the boiler system of Fig.12

The fault trees obtained from the synthesis phase are analysed in the analysis phase. Both qualitative and quantitative analyses could be performed, and eventually an FMEA is created from these results. Usually synthesised fault trees are large and complex; however, they can be minimised. Therefore, the size and complexity of fault trees are reduced by performing qualitative analysis to obtain MCSs. Complex logical expressions are reduced by applying logical rules. As already discussed in section 2, there are many methods available to analyse fault trees. HiP-HOPS uses a version of the MICSUP (Pande et al., 1975) as its primary minimal cut set generating algorithm. Once the MCSs have been obtained, they are used in the quantitative analysis to obtain the probability of the system failure based on basic components failure probability.

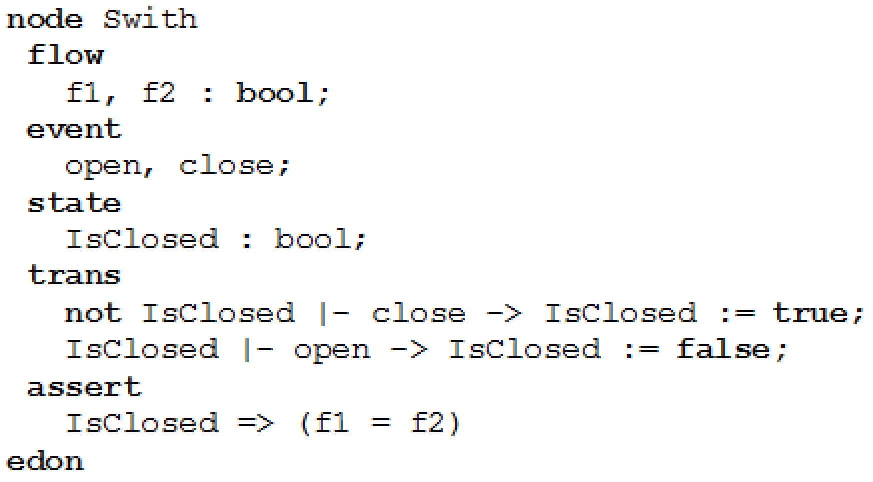
In addition to the quantitative analysis, in the last step a further qualitative analysis is performed to generate an FMEA table. The FMEA table shows the direct connections between component failures and system failures. As a result, from an FMEA table it is possible to understand the effects of a component failure to the whole system and also the likelihood of that failure. Although a conventional FMEA only can show the direct effects of the single failure events on the system, the FMEA generated by HiP-HOPS can also show the possible effects of a failure event if it occurs in combination with other failure events?

4.3 AltaRica

AltaRica is a high level description language (Arnold et al., 2000; Point & Rauzy, 1999) based on finite state machines designed to model both functional and failure behaviour of complex systems. It can represent systems as hierarchies of components and subcomponents and model both state and event like State-Event fault trees. Once a system has been modelled in AltaRica, it can be analysed by external tools and methods, e.g., the generation of fault trees, Petri nets, model-checking etc.(Bieber, Castel, & Seguin, 2002).

In AltaRica, components are represented as nodes, and each node possesses a number of state and variables (either state variables or flow variables). The number of state and flow variable are discrete. The values of the state variables are local to the node they are in, and value of a state variable changes when an event occurs, i.e., events are triggering state transitions, thus changing the values of state variables. Flow variables are visible both locally and globally, and are therefore used to provide interfaces to the nodes.

Each basic component is described by an interfaces transition system, containing the description of the possible events, possible observations, possible configurations, mappings of what observations are linked to which configurations, and what transitions are possible for each configuration. A classic example of a component (electrical switch) in AltaRica is shown in Fig. 19.



```
node Swith
 flow
   f1, f2 : bool;
 event
   open, close;
 state
   IsClosed : bool;
 trans
   not IsClosed |- close -> IsClosed := true;
   IsClosed |- open -> IsClosed := false;
 assert
   IsClosed => (f1 = f2)
edon
```

Fig 19. Node Example in AltaRica: Switch (Point & Rauzy, 1999)

The behaviour of a component (node) is defined through assertions and transitions. Assertions specify restrictions over the values of flow and state variables whereas transitions determine causal relations between state variables and events, consisting of a single trigger and a guard condition which put constraint on the transition; guards are basically some assertions over flow and state variables. In the above example, the values of the state variable (*IsClosed)* of the switch could be either true or false and the events *open* and *close* could trigger the transitions between these states. The component has two flow variables: *f1* and *f2,* the assertions over those two variables species that the power on both terminal of the switch will be same when the switch is closed.

After defining the nodes, they can be organized hierarchically. The top-level node represents the system itself, and it consists of other lower-level nodes. Nodes can communicate either through interfaces or through event dependencies. The first process is done by specifying assertions over interfaces and the second one is done by defining a set of broadcast synchronisation vectors. These broadcast synchronisation vectors allow events in one node or component to be synchronised with those in another. These vectors can contain question marks to indicate that an event is not obligatory (e.g., a bulb cannot turn off in response to a power cut if it is already off). Additional constraints can be applied to the vectors to indicate that certain combinations or numbers of events must occur, particularly in the case of these ‘optional’ events, e.g., that at least one of a number of optional events must occur, or that k-out-of-n must occur etc.

Different variants of AltaRica have been designed. The primary difference between the variants is how the variables are updated after firing of transitions. In the first version (Arnold et al., 2000; Point & Rauzy, 1999), variables are updated by solving constraints, thus consuming too many resources. Therefore, this approach is not scalable for industrial application although it is very powerful. To make AltaRica capable of assessing industrial scale systems, a second version, AltaRica Data-Flow (Boiteau, Dutuit, Rauzy, & Signoret, 2006; Rauzy, 2002) was introduced where variables are updated by propagating values in a fixed order, and the order is determined at compile time. This approach takes fewer resources than the first approach, however, it cannot naturally model bidirectional flows through a network, cannot capture located synchronisation, and faces difficulties in modelling looped systems. The current version of AltaRica is AltaRica 3.0 (Batteux, Prosvirnova, Rauzy, & Kloul, 2013). It improves the expressive power of the second version without reducing the efficiency of assessment algorithms. The main improvement is that it defines the system model as a Guarded Transition Systems (GTS) which allows analysts to model systems consisting of loops, and can easily model bidirectional flows. To analyse AltaRica models, AltaRica 3.0 provides a set of tools, e.g., Fault Tree generator, Markov chain generator, stochastic and stepwise simulator. However, it still lacks capability for capturing dynamic or temporal ordering of events in the generated Fault Trees. Researchers like Bieber et al. (2002) and Li & Li (2014) have proposed methodologies to directly generate classical fault trees from AltaRica models. On the other hand, Bozzano et al. (2011) have proposed a novel approach to translate AltaRica models into an extended version of NuSMV, which could be in turn analysed to obtain classical fault trees. An approach to generate MCSs from AltaRica models was shown in (Griffault, Point, Kuntz, & Vincent, 2011).

4.4 Formal Safety Analysis Platform—New Symbolic Model Verifier

The Formal Safety Analysis Platform FSAP/NuSMV-SA (Bozzano & Villafiorita, 2003, 2007) consists of a set of tools including a graphical user interface tool, FSAP, and an extension of model checking engine NuSMV. The aim of this platform is to support formal analysis and safety assessment of complex systems. This platform allows the user to inject particular failure modes into the system (either nominal mode or degraded mode of the system) and then observe the effects of that failure on the system behaviour. The primary goal of the FSAP/NuSMV-SA toolset is to verify if a system model is satisfying its safety requirements; however, it is capable of performing different types of safety analyses, e.g., automatic fault tree generation.

The components of the FSAP/NuSMV-SA platform are shown in Fig. 20. As seen in the figure, it has different modules to perform different tasks. The central module of the platform is the SAT Manager which can control the other modules of the platform. It stores all the information related to safety assessment and verification which includes references to system model, extended system model, failure modes, safety requirements and analyses. System models are described as finite state machines with the NuSMV language in the Model Capturing module as plain text. This model can be a formal safety model or a functional system model and the user has the flexibility to use their preferred text editor to design or edit the system model. The Failure Mode Editor and the Fault Injector modules allow the user to inject failure modes in the system model to create an extended system model. The expressions of the failure modes can be stored in a library to provide greater degree of reusability. The system model is then augmented with safety requirements in the Safety Requirement Editor. Temporal logic is used to express the safety requirements and can also be stored in a library to facilitate future reuse. The Analysis Task Manager defines the analysis tasks that are required to be performed, i.e., specification of the analyses. The next step is to assess the annotated system model against its functional safety requirements. This task is done based on model checking technique in the NuSMV-SA Model Checker module. This module also generates counter examples and safety analysis results by means of fault trees etc. The Result Extraction and Displayers modules process all the results generated by the platform and present to the user. The fault trees can be viewed in the displayer that is embedded in the platform or using commercial tools, and counter examples can be viewed in a textual or graphical or tabular fashion etc.

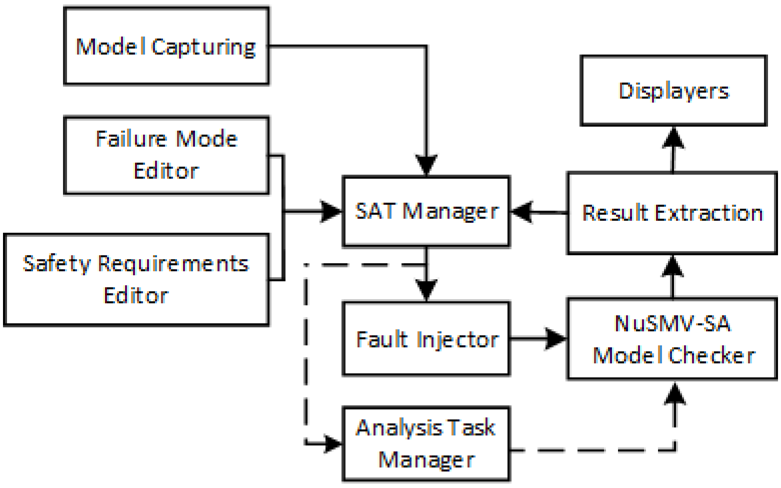


Fig 20. Components of FSAP/NuSMV-SA Platform (Sharvia et al., 2015)

In the FSAP/NuSMV-SA, fault trees are generated step by step. At first a reachability analysis is performed by starting with the initial states of the system model and traversing (by iteratively creating a set of successor states) the system state chart until a state is reached in which the top event of the fault tree has occurred. The outcome of this process is a set of all states where failures of components occur. The minimal cut sets are obtained from these set of states by extracting only the failure variables from these states and writing the expressions for those failure modes based on the information obtained from the state machine. As standard FTs are combinatorial this process ignores dynamic information stored in the system model. However, to preserve the dynamic information NuSMV-SA can add traces with each cut set describing how the failure can occur. Another way of doing the dynamic analysis is to perform the ordering analysis on minimal cut sets (Walker et al., 2008).

Although the FSAP/NuSMV-SA platform provides multiple functionality, it does also have some limitations, especially in handling fault trees. Fault trees generated by this toolset show the relation between top events and basic events; however, how the faults are propagated through the different level of the system model is not shown which could make the fault trees less visibly understandable for complex systems. It provides the means to perform qualitative FTA, but it does not have the ability to perform probabilistic evaluation of FTs. Like other state machine based approaches, this platform also suffers from state space explosion while modelling large or complex systems

4.5 AADL

Architecture Analysis and Design Language (AADL) is a domain-specific language standardised by the International Society of Automotive Engineers (SAE) for the specification and analysis of hardware and software architectures of performance-critical real-time systems (SAE, 2012). It has the capability to represent a system as a collection of software components mapped onto an execution platform. It provides a standard set of component abstractions, such as 'thread', 'thread group', 'process', 'data', and 'subprogram' for software components; 'processor', 'memory', 'device' and 'bus' for hardware components; and 'system' as a composite component (Feiler, Gluch, & Hudak, 2006). AADL supports different types of interaction between components, for example events and dataflows, and the interactions between hardware and software components are defined through binding. Moreover, the language is extensible through customised "annexes". AADL Error Model Annex supports specification of fault and failure information, i.e., it allows to annotate original AADL architecture models with failure related information, thus enabling dependability assessments of systems based on these annotated models.

Several model transformation based methods have been proposed so far to support various analyses based on AADL models. Sokolsky, Lee, & Clark (2006) have proposed a method for schedulability analysis by translating AADL models into the real time process algebra known as the Algebra of Communicating Shared Resources (ACSR) (Lee, Bremond-Gregoire, & Gerber, 1994). The first approach to automatically translate an AADL error model into a standard fault tree was proposed by Joshi, Vestal, & Binns (2007). Dehlinger & Dugan (2008) have extended this work by automatically translating AADL models into dynamic fault trees. At the same time, Sun, Hauptman, & Lutz (2007) have proposed an approach to generate product-line fault tree from AADL models. Li, Zhu, Ma, & Xu (2011) have also proposed a method to translate AADL models into fault trees. A model transformation framework has been proposed in

(Mian, Bottaci, Papadopoulos, & Biehl, 2012), where AADL models are transformed into a model compatible with HiP-HOPS tool, and this new model is then synthesised by the HiP-HOPS tool to generate fault trees and FMEA. A number of researchers have integrated AADL and AltaRica Dataflow. For example, Mokos, Katsaros, Bassiliades, Vassiliadis, & Perrotin (2008) have presented a method for transforming the AADL model to AltaRica model. To perform dependability analysis, AADL models have also been translated into stochastic Petri Nets (e.g. in (Rugina, Kanoun, & Kaâniche, 2007, 2008)) and temporal fault trees (e.g. in (Mahmud & Mian, 2013)). A detail description of different analyses from AADL models is available in (Delange, Feiler, Gluch, & Hudak, 2014) and a list of tool to support such analyses are shown in Table 4. Although the AADL models can be used to model dynamic behaviour of systems, it suffers from state space explosion problem while modelling large and complex systems.

Table 4. Summary of the MBDA approaches

Approaches	Features	Results	Tool Support	Limitations
FPTN	Formal notation for system behaviour. Temporal Extension.	CFT	SSAP Toolset (Fenelon & McDermid, 1993)	Lack of full automation. No provision for quantitative analysis with uncertain data.
HiP-HOPS	Automated safety and reliability analysis. Temporal Extension. Multi objective Architecture optimisation. Semi-automated ASIL allocation.	SFT Pandora TFT FMEA	HiP-HOPS tool (Papadopoulos, 2012)	Lack of automated dynamic safety analysis. No provision for quantitative analysis with uncertain data.
AltaRica	Formal Verification. Timed automata and GTS extension.	SFT	OpenAltaRica (IRT SystemX, 2016)	State explosion. No provision for quantitative analysis with uncertain data.
FSAP/NuSMV	Formal verification. Library of failure modes.	SFT	FSAP/NuSMV-SA (FBK, 2006)	State explosion. Fault tree structure. Lack of dynamic analysis. No provision for quantitative analysis with uncertain data.
AADL	Domain concepts with strong semantics. Error model annex for dependability analysis.	SFT DFT TFT GSPN	OSATE (OSATE, 2016); AADL Inspector (Ellidiss Software, 2016); Ocarina (Open AADL, 2016)	State explosion. No provision for quantitative analysis with uncertain data.

4.6 Other Approaches

The section briefly describes two other approaches, where fault tree analysis could be used as a means for system dependability analysis.

Deductive Cause Consequence Analysis or DCCA (Güdemann, Ortmeier, & Reif, 2007; Ortmeier, Reif, & Schellhorn, 2005) is a formal method for safety analysis. It determines whether a given component fault is the cause of a system failure or not by using mathematical methods. It is a formal generalization of FMEA and FTA, more formal than FTA and more expressive than FMEA. DCCA represents the system model as finite state automata with temporal semantics using Computational Tree Logic (CTL). It assumes that all the basic component failure modes are available, and then defines a set of temporal properties that indicate whether a certain combination of component failure modes can lead to the system failure. This property is known as the criticality of a set of failure modes which are analogous to cut sets of classical fault trees. Similar to FTA, DCCA aims at determining the minimal critical sets of failure modes which are necessary and sufficient to cause the top event (system failure) and therefore FTA may be considered as a special case of DCCA. Deductive Failure Order Analysis (Güdemann, Ortmeier, & Reif, 2008), an extension of DCCA enables it to deduce temporal ordering information of critical sets from DCCA. In this extension, Pandora style temporal gates like PAND and SAND are used to capture temporal behaviour. Temporal logic laws are also provided to make the temporal ordering deduction process automated.

Safety Analysis Modelling Language (SAML) (Güdemann & Ortmeier, 2010) is a tool independent modelling framework that can construct system models with both deterministic and probabilistic behaviour. It utilises finite state automata with parallel synchronous execution capability with discrete time steps to describe a system model consisting of hardware and/or software components, environmental conditions etc. In the state automata, transitions can be defined both as probabilistic and non-deterministic. From a single SAML model both qualitative and quantitative analysis can be performed.

SAML models can be transformed automatically to the input format of other model based safety analysis techniques without changing the architecture of the systems. Therefore SAML can work as an intermediate language for MBDA techniques, i.e., if models designed in any other higher-level language can be converted to SAML models (extended system models) then the resultant models can be transformed to input format of other targeted analysis tools, and, thereby analysed with all targeted tools. A method of transforming SAML models into the input language of probabilistic model checker PRISM (Kwiatkowska, Norman, & Parker, 2011) was represented in (Güdemann & Ortmeier, 2011). In the same work, the ways of transforming SAML modules to NuSMV modules were also demonstrated.

5. Discussion and Future Outlook for MBDA

It is a priority for system analysts and engineers to ensure the dependability of safety-critical systems by identifying the potential risks they pose as early as possible and then minimising the likelihood of these risks. The most widely used approach for dependability analysis is the fault tree analysis (FTA). However, the manual nature of the FTA makes the system analyses process time consuming and expensive. At the same time, as the analyses are performed on informal system models, these processes can result in inconsistencies and discrepancies leading to inaccurate evaluation of system dependability.

Model based dependability analysis paradigm overcomes the above problems by simplifying the dependability analysis process by automatically synthesising dependability related data from system models to generate dependability analysis artefacts such as fault trees and FMEAs. This means that the system analysts and risk modellers can apply MBDA approaches on the early design models of the systems as part of model-based design process. As the system models tend to change rapidly in the early design phases, analysts can take advantage of the MBDA approaches and perform analyses on the formal models iteratively to generate more results, thus refining and synchronising the dependability analysis results with the evolving system designs. In addition to that, in the design process, analysts can reuse parts of an existing system model, or libraries of previously analysed components. All these features provided by the MBDA approaches would result in more consistent and complete results and a significant reduction in design time and efforts.

Although the MBDA tools and techniques help to simplify and automate the dependability analysis process, there exist many developments and challenges in this area. From the review, it is observed that MBDA approaches tend to follow two different paradigms. The first group of approaches (FPTN and HiP-HOPS) focus more on the automatic construction of predictive system analyses such as fault trees or FMEAs. Techniques in these group generally use a dedicated system model developed solely for the dependability analysis. By using dedicated dependability analysis model, the analysts can avoid unnecessary complexity of augmenting the original system model. However, to use a dedicated model the analysts have to put extra effort to create the model and additional effort is required to maintain consistency between the original system model and the dependability analysis model. These approaches are typically compositional, meaning that system-level failure analyses can be generated from component-level failure logic and the topology of the system. This compositionality lends itself well to automation and reuse of component failure models across applications, thus allowing rapid evaluation of speculative changes to the system model. Due to this efficient nature of these approaches, the analysts can perform valuable analysis on the early design models when concrete system details are scarce. A limitation of these approaches are that they have limited ability to perform dynamic dependability analysis because of their reliance on static analysis approaches such as FTA for the purpose of dependability analysis.

The second group of approaches (AltaRica and FSAP-NuSMV) focus on behavioural simulation for automatic verification of system dependability properties. The primary strengths of these approaches are their capability to perform dynamic dependability analysis and facilitate automated formal verification. These approaches generally use state-event based formalisms and work by injecting possible faults into simulations based on executable. As the analysis is performed on the extended version of the original system model (not on a dedicated model), extra effort for maintaining consistency between models is no more needed. However, the use of the extended system model may impose constraints on the dependability analysis as explained in Lisagor, Kelly, & Niu (2011). In addition to that, as the executable design models are created in the later stages of the design process, the design changes are costly to implement in these stages. As a result, the analysis results often lose the opportunity to drive the design process. Moreover, dependability artefacts such as fault trees generated by these approaches tend to have flat structure, which may cause difficulties for analysts to visually understand these results. As these approaches work on state-event based models of the system, they suffer from state space explosion while exploring huge state space of complex systems. This also makes the analysis computationally expensive. Therefore, future research associated with these approaches are likely to concern with the improvement of the power and time complexity of the tools and techniques in the context of large and complex system models.

In order to consolidate both architectural and behavioural information of the system in a single model to support multiple analyses, the large-scale architectural annotation languages (ADLs), such as AADL in the aerospace domain or EAST-ADL (EAST-ADL Association, 2014) in the automotive domain has increasingly being adopted. This trend has led to a number of model transformations between MBDA approaches and ADLs. Future trends are likely to leading to more robust integrations between different existing MBDA approaches so that different strengths (e.g. dependability analysis and model checking capability) of the existing approaches can be utilised in a complementary manner. The state-space explosion problem, which is inherently part of state-based techniques, can be addressed with abstraction techniques (although this is a largely complex subject in itself).

In terms of dynamic dependability analysis, less development has happened in the area of MBDA. As mentioned earlier, there exist different tools to analyse dynamic fault trees or other dynamic extensions of static fault trees for dynamic dependability analysis. However, MBDA approaches are limited in their ability to generate dynamic extensions of SFTs by synthesising system models. As there are enormous tool supports (e.g. Galileo (Dugan, Sullivan, & Coppit, 2000), RAATSS (Chiacchio, 2012, Manno et al., 2014), DFTSim (Boudali et al., 2009) etc.) available for dynamic fault tree analysis, future trends are likely to leading to development of expert systems so that MBDA approaches could be utilised to generate dynamic extensions of fault trees. Given a qualitative dynamic fault tree (generated automatically by MBDA approaches), it can be imported into tools like Galileo (Dugan et al., 2000), RAATSS (Chiacchio, 2012, Manno et al., 2014), and MatCarloRe (Manno, Chiacchio, Compagno, D'Urso, & Trapani, 2012) etc. to perform the quantitative analysis.

One common feature of all the MBDA techniques is that they perform quantitative analysis based on fixed values of failure data of system components, and hence take it as guaranteed that the precise failure data of components are always available. As already mentioned, in many situations the availability of the failure data of system components could not be guaranteed. In such situations, the MBDA approaches would not be able to perform quantitative dependability analysis due to the unavailability of the failure data. As revealed by the review, many independent researches have been performed and many standalone approaches have been developed to address the issue of uncertain quantitative data in the quantitative analysis of classical and dynamic fault trees. Therefore, it is worthwhile perform research to develop an expert system by integrating uncertainty quantification approaches with MBDA approaches, thus allowing automatic dependability analysis of complex systems under the conditions of uncertainty.

In terms of application, the state-of-the-art MBDA approaches are presently applied during design phase for predictive analysis of systems. To do so, these approaches require a priori knowledge of the possible system configurations. However, technological advancement has brought systems like Cyber Physical Systems (CPS) and Internet of Things (IoT), which are loosely connected systems, i.e., smaller systems may connected together to form a configuration and this configuration dissolves after a certain period of time to give place to other configurations. Hence, dependability of such systems cannot be ensured using the currently available approaches due to the unknown number of potential system configurations. This has open new avenues for further research to develop expert systems by combining MBDA approaches with other soft computing approaches for the assurance of dependability of such open systems. One possible avenue worthy of further research is the improvement of the MBDA approaches to perform real time analysis of systems—though it will complicate the analysis process and affect the scalability of the approaches. In real time environment, the approaches will have to handle a huge amount of data and process them to make meaningful decision to improve dependability of the systems. All the data does not necessarily need to be analysed online, offline analysis of vast amount data will also be needed. To obtain such capability, software tools and expert systems implementing MBDA can be empowered with concepts like data mining and machine learning etc.

6. Conclusion

This paper provided an overview of fault tree analysis. Although FTA is a highly successful and widely-used method for dependability analysis of wide variety of systems, it does have a number of limitations, such as an inability to model sequence- or time-dependent dynamic behaviour and to perform quantitative analysis with uncertain failure data. In addition to that, even where software tool support exists for FTA, it requires a lot of manual efforts to create and analyse fault trees. Different extensions of standard fault trees have been proposed to overcome some of the limitations. Many of these extensions are also discussed in this paper.

Over the past twenty years, researchers have made continuous efforts to simplify the dependability analysis process by automatically synthesising dependability related data from system models. This has led to the emergence of the field of model-based dependability analysis (MBDA). MBDA has attracted significant interest from academia and industry. Several tools and techniques have been developed to support MBDA. MBDA techniques allow dependability models and analyses to be automatically synthesised from system engineering models. This offers important advantages, not least the reduction in both effort and potential for error, and supports a more iterative design process via automatic synthesis of failure models. This paper reviewed a number of prominent MBDA techniques, exploring their working mechanism, strengths, limitations, and recent developments.

Closing this paper, it is emphasised that MBDA is an emerging field which provides tools and techniques to automate the dependability analysis process, but there are many future developments and challenges remain in this area. This paper has discussed some of these challenges and provided directions for future research. First of all, as most of the MBDA approaches lack in their capability to perform dynamic dependability analysis, it is believed that it is both theoretically and practically important to explore possible ways to develop expert systems for model based dynamic analysis of systems. It has also been emphasised that more research should be done to develop an expert system by integrating uncertainty quantification approaches with MBDA approaches so that model based dependability analysis of complex systems could be performed automatically under the conditions of uncertainty. Secondly, it has been predicted that the future research may lead to more integrated approach, where different strengths of the existing state-of-the-art MBDA approaches will be used in a complementary manner. Finally, emphasise has been put on to perform further research to empower the existing MBDA techniques with new concepts such as machine learning, data mining to make them capable to perform dependability analysis of large and complex open systems.

References

Adler, R., Domis, D., Höfig, K., Kemmann, S., Kuhn, T., Schwinn, J.-P., & Trapp, M. (2011). Integration of Component Fault Trees into the UML. In *Workshops and Symposia at MODELS* (pp. 312–327).

Aizpurua, J. I., & Muxika, E. (2013). Model-Based Design of Dependable Systems : Limitations and Evolution of Analysis and Verification Approaches. *International Journal on Advances in Security*, *6*(1), 12–31.

Åkerholm, M., Carlson, J., Fredriksson, J., Hansson, H., Håkansson, J., Möller, A., Pettersson, P., & Tivoli, M. (2007). The SAVE approach to component-based development of vehicular systems. *Journal of Systems and Software*, *80*(5), 655–667.

ALD. (2014). ALD RAM Commander. In: Available online at: http://aldservice.com/reliability-products/rams-software.html

Anand, A., & Somani, A. K. (1998). Hierarchical analysis of fault trees with dependencies, using decomposition. In *Proceedings of Annual Reliability and Maintainability Symposium,* (pp. 69–75). doi:10.1109/RAMS.1998.653591

Andrews, J. D. (1998). Tutorial: Fault Tree Analysis. In *Proceedings of the 16th International System Safety Conference*. In: Available online at: http://www.fault-tree.net/papers/andrews-fta-tutor.pdf

Arnold, A., Point, G., Griffault, A., & Rauzy, A. (2000). The AltaRica formalism for describing concurrent systems. *Fundamenta Informaticae*, *40*(2), 109–124.

Arnold, F., Belinfante, A. F. E., Van der Berg, F. I., Guck, D., & Stoelinga, M. I. A. (2013). DFTCalc: A Tool for Efficient Fault Tree Analysis. In *32nd International Conference on Computer Safety, Reliability and Security (SAFECOMP'13)* (pp. 293–301).

Aubry, J.-F., & Brînzei, N. (2015). Stochastic Hybrid Automaton. In *Systems Dependability Assessment* (pp. 105–120).

https://doi.org/10.1002/9781119053996

Auvation. (2016). OpenFTA. In: Available online at: http://www.openfta.com/

Balakrishnan, M., & Trivedi, K. (1995). Componentwise decomposition for an efficient reliability computation of systems with repairable components. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing* (pp. 259–268).

Batteux, M., Prosvirnova, T., Rauzy, A., & Kloul, L. (2013). The AltaRica 3.0 project for model-based safety assessment. In *11th IEEE International Conference on Industrial Informatics (INDIN)* (pp. 741–746). IEEE.

Beccuti, M., Codetta-Raiteri, D., Franceschinis, G., & Haddad, S. (2008). Non deterministic Repairable Fault Trees for computing optimal repair strategy. In *Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools*.

Beeson, S. C. (2002). *Non-coherent Fault Tree Analysis*. Loughborough University, PhD Thesis.

Bieber, P., Castel, C., & Seguin, C. (2002). Combination of fault tree analysis and model checking for safety assessment of complex system. In *Proceedings of the 4th European Depting Conference on Dependable Computing (EDCC)* (pp. 19–31).

Bobbio, A., Portinale, L., Minichino, M., & Ciancamerla, E. (2001). Improving the analysis of dependable systems by mapping fault trees into Bayesian networks. *Reliability Engineering & System Safety*, *71*(3), 249–260. doi:10.1016/S0951-8320(00)00077-6

Boiteau, M., Dutuit, Y., Rauzy, A., & Signoret, J. P. (2006). The AltaRica data-flow language in use: modeling of production availability of a multi-state system. *Reliability Engineering & System Safety*, *91*(7), 747–755.

Boudali, H., Crouzen, P., & Stoelinga, M. (2007a). A compositional semantics for Dynamic Fault Trees in terms of Interactive Markov Chains. In *Automated technology for verification and analysis* (pp. 441–456).

Boudali, H., Crouzen, P., & Stoelinga, M. (2007b). Dynamic Fault Tree analysis using Input / Output Interactive Markov Chains. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (pp. 708–717). Washington DC: IEEE Computer Society.

Boudali, H., Crouzen, P., & Stoelinga, M. (2010). A Rigorous, Compositional, and Extensible Framework for Dynamic Fault Tree Analysis. *IEEE Transactions on Dependable and Secure Computing*, *7*(2), 128–143. doi:10.1109/TDSC.2009.45

Boudali, H., & Dugan, J. B. (2005). A discrete-time Bayesian network reliability modeling and analysis framework. *Reliability Engineering and System Safety*, *87*(3), 337–349. doi:10.1016/j.ress.2004.06.004

Boudali, H., & Dugan, J. B. (2006). A Continuous-Time Bayesian Network Reliability Modeling, and Analysis Framework. *IEEE Transaction on Reliability*, *55*(1), 86–97.

Boudali, H., Nijmeijer, A. P., & Stoelinga, M. (2009). DFTSim: A Simulation Tool for Extended Dynamic Fault Trees. In *42nd Annual Simulation Symposium* (pp. 31–38).

Bozzano, M., Cimatti, A., Lisagor, O., Mattarei, C., Mover, S., Roveri, M., & Tonetta, S. (2011). Symbolic Model Checking and Safety Assessment of Altarica models. In *Proceedings of the 10th International Workshop on Automated Verification of Critical Systems* (Vol. 35).

Bozzano, M., & Villafiorita, A. (2003). Improving System Reliability via Model Checking: The FSAP/NuSMV-SA Safety Analysis Platform. *Computer Safety, Reliability, and Security*, *2788*, 49–62.

Bozzano, M., & Villafiorita, A. (2007). The FSAP/NuSMV-SA Safety Analysis Platform. *International Journal on Software Tools for Technology Transfer (STTT) - Special Section on Advances in Automated Verification of Critical Systems*, *9*(1), 5–24.

BQR. (2015). CARE FTA. In: Available online at: http://www.bqr.com/care/care-dfr-package/care-fta/

Carlson, J., Håkansson, J., & Pettersson, P. (2006). SaveCCM: An Analysable Component Model for Real-Time Systems. *Electronic Notes in Theoretical Computer Science*, *160*, 127–140.

Castaneda, G. A. P., Aubry, J.-F., & Brînzei, N. (2011). Stochastic hybrid automata model for dynamic reliability assessment. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, *225*(1), 28–41. https://doi.org/10.1177/1748006XJRR312

Chiacchio, F. (2012). RAATSS. In: Available online at: http://www.dmi.unict.it/~chiacchio/?m=5&project=raatss

Chiacchio, F., Cacioppo, M., D'urso, D., Manno, G., Trapani, N., & Compagno, L. (2013). A Weibull-based compositional approach for hierarchical dynamic fault trees. *Reliability Engineering and System Safety*, *109*, 45–52. doi:10.1016/j.ress.2012.07.005

Chiacchio, F., D'Urso, D., Compagno, L., Pennisi, M., Pappalardo, F., & Manno, G. (2016). SHyFTA, a Stochastic Hybrid Fault Tree Automaton for the modelling and simulation of dynamic reliability problems. *Expert Systems with Applications*, *47*, 42–57. https://doi.org/10.1016/j.eswa.2015.10.046

Codetta-Raiteri, D. (2005). The Conversion of Dynamic Fault Trees to Stochastic Petri Nets, as a case of Graph Transformation. *Electronic Notes in Theoretical Computer Science*, *127*(2), 45–60.

Codetta-Raiteri, D., Franceschinis, G., Iacono, M., & Vittorini, V. (2004). Repairable fault tree for the automatic evaluation of repair policies. In *International Conference on Dependable Systems and Networks* (pp. 659–668).

Contini, S., Cojazzi, G. G. M., & Renda, G. (2008). On the use of non-coherent fault trees in safety and security studies. *Reliability Engineering and System Safety*, *93*(12), 1886–1895. doi:10.1016/j.ress.2008.03.018

Dehlinger, J., & Dugan, J. B. (2008). Analyzing dynamic fault trees derived from model-based system architectures. *Nuclear Engineering and Technology*, *40*(5), 365–374. doi:10.5516/NET.2008.40.5.365

Delange, J., Feiler, P., Gluch, D., & Hudak, J. J. (2014). *AADL Fault Modeling and Analysis Within an ARP4761 Safety Assessment,* Technical Report, CMU/SEI-2014-TR-020.

Dugan, J. B., Bavuso, S. J., & Boyd, M. A. (1992). Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, *41*(3), 363–377. doi:10.1109/24.159800

Dugan, J. B., Bavuso, S. J., & Boyd, M. A. (1993). Fault trees and Markov models for reliability analysis of fault-tolerant digital systems. *Reliability Engineering & System Safety*, *39*(3), 291–307.

Dugan, J. B., Sullivan, K. J., & Coppit, D. (2000). Developing a low-cost high-quality software tool for dynamic fault-tree analysis. *IEEE Transactions on Reliability*, *49*(1), 49–59. doi:10.1109/24.855536

Dugan, J. B., Venkataraman, B., & Gulati, R. (1997). DIFtree: a software package for the analysis of dynamic fault tree models. In *Proceedings of Annual Reliability and Maintainability Symposium* (pp. 64–70).

EAST-ADL Association (2014). EAST-ADL V2.1.12 specification. In: Available online at: http://www.east-adl.info/Specification.html

Edifor, E., Walker, M., & Gordon, N. (2012). Quantification of Priority-OR Gates in Temporal Fault Trees. In F. Ortmeier & P. Daniel (Eds.), *Computer Safety, Reliability, and Security SE - 9* (Vol. 7612, pp. 99–110). Springer Berlin Heidelberg.

Edifor, E., Walker, M., & Gordon, N. (2013). Quantification of Simultaneous-AND Gates in Temporal Fault Trees. In W. Zamojski, J. Mazurkiewicz, J. Sugier, T. Walkowiak, & J. Kacprzyk (Eds.), *New Results in Dependability and Computer Systems SE - 13* (Vol. 224, pp. 141–151). Springer International Publishing.

Edifor, E., Walker, M., Gordon, N., & Papadopoulos, Y. (2014). Using Simulation to Evaluate Dynamic Systems with Weibull or Lognormal Distributions. In *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX* (pp. 177–187).

Ejlali, A., & Miremadi, S. G. (2004). FPGA-based Monte Carlo simulation for fault tree analysis. *Microelectronics Reliability*, *44*(6), 1017–1028.

Ellidiss Software. (2016). AADL Inspector. In: Available online at: http://www.ellidiss.com/downloads/

EPRI. (2013). EPRI CAFTA. In: Available online at: http://www.epri.com/abstracts/Pages/ProductAbstract.aspx?ProductId=000000000001015514

Ericson, C. A. (1999). Fault Tree Analysis – a history. In *Proceedings of the 17th International System Safety Conference* (pp. 1–9).

ESSaRel. (2005). Embedded Systems Safety and Reliability Analyser,. *The ESSaRel Research Project*. In: Available online at: http://essarel.de

FBK. (2006). FSAP/NuSMV-SA. In: Available online at: https://es-static.fbk.eu/tools/FSAP/

Feiler, P. H., Gluch, D. P., & Hudak, J. J. (2006). *The Architecture Analysis & Design Language (AADL): An Introduction*. Technical Report, CMU/SEI-2006-TN-011, Carnegie Mellon University.

Feiler, P. H., Lewis, B. A., & Vestal, S. (2006). The SAE Architecture Analysis & Design Language (AADL) a Standard for Engineering Performance Critical Systems. In *Proceedings of the IEEE Conference on Computer Aided Control Systems Design* (pp. 1206–1211). Munich: IEEE. doi:10.1109/CACSD-CCA-ISIC.2006.4776814

Fenelon, P., & McDermid, J. A. (1993). An Integrated Toolset For Software Safety Analysis. *Journal of Systems and Software*, *21*(3), 279–290.

Ferdous, R., Khan, F., Veitch, B., & Amyotte, P. R. (2009). Methodology for computer aided fuzzy fault tree analysis. *Process Safety and Environmental Protection*, *87*(4), 217–226.

Förster, M., & Kaiser, B. (2006). Increased efficiency in the quantitative evaluation of state/event fault trees. *IFAC Proceedings Volumes*, *39*(3), 255–260.

Fussel, J. B., & Vesely, W. E. (1972). A new methodology for obtaining cut sets for fault trees. *Transactions of the American Nuclear Society*, *15*(1), 262–263.

Fussell, J. B., Aber, E. F., & Rahl, R. G. (1976). On the Quantitative Analysis of Priority-AND Failure Logic. *IEEE Transactions on Reliability*, *R-25*(5), 324–326.

German, R., & Mitzlaff, J. (1995). Transient Analysis of Deterministic and Stochastic Petri Nets with TimeNET. In *Proceedings of the 8th International Conference on Computer Performance Evaluation, Modelling Techniques, and Tools and MMB* (pp. 209–223). Springer-Verlag.

Gluchowski, P. (2007). Duration Calculus for Analysis of Fault Trees with Time Dependencies. In *2nd International Conference on Dependability of Computer Systems* (pp. 107–114).

Gmytrasiewicz, P., Hassberger, J. A., & Lee, J. C. (1990). Fault tree based diagnostics using fuzzy logic. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *12*(11), 1115–1119.

GRIF. (2016). GRIF-Workshop. In: Available online at: http://grif-workshop.com/

Griffault, A., Point, G., Kuntz, F., & Vincent, A. (2011). *Symbolic computation of minimal cuts for AltaRica models*. Research Report RR-1456-11.

Grunske, L. (2006). Towards an Integration of Standard Component-Based Safety Evaluation Techniques with SaveCCM. In C. Hofmeister, I. Crnkovic, & R. Reussner (Eds.), *Proceedings of the Second International Conference on Quality of Software Architectures (QoSA'06)* (Vol. 4214, pp. 199–213). Berlin, Heidelberg: Springer Berlin Heidelberg.

Grunske, L., & Han, J. (2008). A Comparative Study into Architecture-Based Safety Evaluation Methodologies Using AADL's Error Annex and Failure Propagation Models. In *11th IEEE High Assurance Systems Engineering Symposium* (pp. 283–292). Ieee.

Grunske, L., & Kaiser, B. (2005a). An automated dependability analysis method for COTS-based systems. In *COTS-Based Software Systems* (pp. 178–190).

Grunske, L., & Kaiser, B. (2005b). Automatic Generation of Analyzable Failure Propagation Models from Component-Level Failure Annotations. In *Fifth International Conference on Quality Software(QSIC 2005).* (pp. 117–123).

Grunske, L., Kaiser, B., & Papadopoulos, Y. (2005). Model-Driven safety evaluation with state-event-based component failure annotations. In G. T. Heineman, I. Crnkovic, H. W. Schmidt, J. A. Stafford, C. Szyperski, & K. Wallnau (Eds.), *Proceedings of the 8th international conference on Component-Based Software Engineering (CBSE'05)* (Vol. 3489, pp. 33–48). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/b136248

Grunske, L., Kaiser, B., & Reussner, R. H. (2005). Specification and evaluation of safety properties in a component-based software engineering process. In *Component-Based Software Development for Embedded Systems* (pp. 249–274).

Güdemann, M., & Ortmeier, F. (2010). A Framework for Qualitative and Quantitative Formal Model-Based Safety Analysis. In *Proceedings of 12th International Symposium on High-Assurance Systems Engineering (HASE)* (pp. 132–141).

Güdemann, M., & Ortmeier, F. (2011). Towards Model-driven Safety Analysis. In *3rd International Workshop on Dependable Control of Discrete Systems (DCDS)* (pp. 53–58).

Güdemann, M., Ortmeier, F., & Reif, W. (2007). Using Deductive Cause-Consequence Analysis (DCCA) with SCADE. In *26th International Conference in Computer Safety, Relaibility, and Security* (pp. 465–478).

Güdemann, M., Ortmeier, F., & Reif, W. (2008). Computation of ordered minimal critical sets. In *Proceedings of the*

7th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 08).

Guimarees, A. C. F., & Ebecken, N. F. F. (1999). FuzzyFTA : a fuzzy fault tree system for uncertainty analysis. *Annals of Nuclear Energy*, *26*, 523–532.

Gulati, R., & Dugan, J. B. (1997). A Modular Approach for Analyzing Static and Dynamic Fault Trees. In *Proceedings of the Annual Reliability and Maintainability Symposium* (pp. 57–63). doi:10.1109/RAMS.1997.571665

Huang, C.-Y., & Chang, Y.-R. (2007). An improved decomposition scheme for assessing the reliability of embedded systems by using dynamic fault trees. *Reliability Engineering & System Safety*, *92*(10), 1403–1412. doi:10.1016/j.ress.2006.09.008

Idaho National Laboratory (2016). SAPHIRE - Systems Analysis Programs for Hands-on Integrated Reliability Evaluations. In: Available online at: https://saphire.inl.gov/

IRT SystemX. (2016). OpenAltaRica. In: Available online at: http://openaltarica.fr/getting-started/

ISO. (2011). *ISO 26262: Road vehicles - functional safety.* Geneva, Switzerland.

Isograph. (2016). Isograph FaultTree+. In: Available online at: http://www.isograph.com/software/reliability-workbench/fault-tree-analysis/

ITEM Software (2016). ITEM ToolKit. In: Available online at: http://www.itemsoft.com/item_toolkit.html

Joshi, A., Heimdahl, M. P. E., Miller, S. P., & Whalen, M. W. (2006). *Model-based Safety Analysis*. NASA Technical Report, NASA/CR-2006-213953. NASA Langley Research Center, Hampton, VA, USA.

Joshi, A., Vestal, S., & Binns, P. (2007). Automatic Generation of Static Fault Trees from AADL Models. In *DSN Workshop on Architecting Dependable Systems*.

Kabir, S. (2016). *Compositional Dependability Analysis of Dynamic Systems with Uncertainty*. University of Hull, PhD Thesis.

Kabir, S., Azad, T., Walker, M., & Gheraibia, Y. (2015). Reliability analysis of automated pond oxygen management system. In *18th International Conference on Computer and Information Technology* (pp. 144–149). https://doi.org/10.1109/ICCITechn.2015.7488058

Kabir, S., Edifor, E., Walker, M., & Gordon, N. (2014). Quantification of Temporal Fault Trees Based on Fuzzy Set Theory. In *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX* (pp. 255–264). Brunów: Springer International Publishing. doi:10.1007/978-3-319-07013-1_24

Kabir, S., Walker, M., & Papadopoulos, Y. (2014). Reliability Analysis of Dynamic Systems by Translating Temporal Fault Trees into Bayesian Networks. In F. Ortmeier & A. Rauzy (Eds.), *Model-Based Safety and Assessment* (Vol. 8822, pp. 96–109). Cham: Springer International Publishing. doi:10.1007/978-3-319-12214-4

Kabir, S., Walker, M., & Papadopoulos, Y. (2015). Quantitative evaluation of Pandora Temporal Fault Trees via Petri Nets. IFAC-PapersOnLine, 48(21), 458–463. doi:http://dx.doi.org/10.1016/j.ifacol.2015.09.569

Kabir, S., Walker, M., Papadopoulos, Y., Rüde, E., & Securius, P. (2016). Fuzzy temporal fault tree analysis of dynamic systems. *International Journal of Approximate Reasoning*, *77*, 20–37. doi:10.1016/j.ijar.2016.05.006

Kaiser, B., Gramlich, C., & Förster, M. (2007). State/event fault trees—A safety analysis model for software-controlled systems. *Reliability Engineering & System Safety*, *92*(11), 1521–1537.

Kaiser, B., Liggesmeyer, P., & Mäckel, O. (2003). A New Component Concept for Fault Trees. In *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS'03)* (Vol. 33, pp. 37–46).

Kwiatkowska, M., Norman, G., & Parker, D. (2011). PRISM 4.0: verification of probabilistic real-time systems. In *Proceedings of the 23rd international conference on Computer aided verification (CAV'11)* (pp. 585–591). Springer-Verlag.

Lee, I., Bremond-Gregoire, P., & Gerber, R. (1994). A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems. Proceedings of the IEEE, 82(1), 158–171.

Leveson, N. G. (1995). Safeware: System Safety and Computers. Addison-Wesley.

Li, S., & Li, X. (2014). Study on generation of fault trees from Altarica models. Procedia Engineering, 80, 140–152. doi:10.1016/j.proeng.2014.09.070

Li, Y. F., Huang, H. Z., Liu, Y., Xiao, N., & Li, H. (2012). A new fault tree analysis method : fuzzy dynamic fault tree analysis. Eksploatacja I Niezawodnosc-Maintenance and Reliability, 14(3), 208–214.

Li, Y. F., Mi, J., Liu, Y., Yang, Y. J., & Huang, H. Z. (2015). Dynamic fault tree analysis based on continuous-time Bayesian networks under fuzzy numbers. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, 1–12. doi:10.1177/1748006X15588446

Li, Y., Zhu, Y., Ma, C., & Xu, M. (2011). A method for constructing fault trees from AADL models. In Autonomic and Trusted Computing, Lecture Notes in Computer Science (Vol. 6906, pp. 243–258). doi:10.1007/978-3-642-23496-5_18

Liang, G.-S., & Wang, M.-J. J. (1993). Fuzzy fault-tree analysis using failure possibility. Microelectronics Reliability, 33(4), 583–597.

Lin, C., & Wang, M. J. (1997). Hybrid fault tree analysis using fuzzy sets. Reliability Engineering and System Safety, 58(3), 205–213.

Lisagor, O., Kelly, T., & Niu, R. (2011). Model-based safety assessment: Review of the discipline and its challenges. *9th International Conference on Reliability, Maintainability and Safety*, 625–632. https://doi.org/10.1109/ICRMS.2011.5979344

LOGAN. (2016). LOGAN Fault and Event Tree Analysis. In: Available online at: http://loganfta.com/

Mahmood, Y. A., Ahmadi, A., Verma, A. K., Srividya, A., & Kumar, U. (2013). Fuzzy fault tree analysis : a review of concept and application. *International Journal of System Assurance Engineering and Management*, *4*(1), 19–32. doi:10.1007/s13198-013-0145-x

Mahmud, N., & Mian, Z. (2013). Automatic generation of Temporal Fault Trees from AADL models. In *Safety, Reliability and Risk Analysis: Beyond the Horizon* (pp. 2741–2749).

Manian, R., Dugan, J. B., Coppit, D., & Sullivan, K. J. (1998). Combining Various Solution Techniques for Dynamic Fault Tree Analysis of Computer Systems. In *Proceedings of Third IEEE International High-Assurance Systems Engineering Symposium* (pp. 21–28). Washington: IEEE. doi:10.1109/HASE.1998.731591

Manno, G., Chiacchio, F., Compagno, L., D'Urso, D., & Trapani, N. (2012). MatCarloRe: An integrated FT and Monte Carlo Simulink tool for the reliability assessment of dynamic fault tree. *Expert Systems with Applications*, *39*(12), 10334–10342. doi:10.1016/j.eswa.2011.12.020

Manno, G., Chiacchio, F., Compagno, L., D'Urso, D., & Trapani, N. (2014). Conception of Repairable Dynamic Fault Trees and resolution by the use of RAATSS, a Matlab toolbox based on the ATS formalism. *Reliability Engineering & System Safety*, *121*, 250–262. https://doi.org/10.1016/j.ress.2013.09.002

Marquez, D., Neil, M., & Fenton, N. (2008). Solving Dynamic Fault Trees using a New Hybrid Bayesian Network Inference Algorithm. In *16th Mediterranean Conference on Control and Automation* (pp. 609–614). Ajaccio: IEEE.

Marsan, M. A., Balbo, G., Conte, G., Donatelli, S., & Franceschinis, G. (1996). *Modeling With Generalized Stochastic Petri Nets*. West Sussex: Wiley. Retrieved from http://www.di.unito.it/~greatspn/GSPN-Wiley/

Marsan, M. A., & Chiola, G. (1987). On Petri nets with deterministic and exponentially distributed firing times. *Advances in Petri Nets, 266*, 132–145.

Merle, G., Roussel, J., Lesage, J., & Bobbio, A. (2010). Probabilistic Algebraic Analysis of Fault Trees With Priority Dynamic Gates and Repeated Events. *IEEE Transactions on Reliability, 59*(1), 250–261.

Merle, G., Roussel, J.-M., & Lesage, J.-J. (2011). Algebraic determination of the structure function of Dynamic Fault Trees. *Reliability Engineering & System Safety, 96*(2), 267–277.

Merle, G., Roussel, J.-M., & Lesage, J.-J. (2014). Quantitative Analysis of Dynamic Fault Trees Based on the Structure Function. *Quality and Reliability Engineering International*, *30*(1), 143–156.

Mian, Z., Bottaci, L., Papadopoulos, Y., & Biehl, M. (2012). System dependability modelling and analysis using AADL and HiP-HOPS. *IFAC Proceedings Volumes (IFAC-PapersOnline), 14*(PART 1), 1647–1652. doi:10.3182/20120523-3-RO-2023.00334

Misra, K. B., & Weber, G. G. (1990). Use of fuzzy set theory for level-I studies in probabilistic risk assessment. *Fuzzy Sets and Systems*, *37*(2), 139–160. doi:10.1016/0165-0114(90)90038-8

Mokos, K., Katsaros, P., Bassiliades, N., Vassiliadis, V., & Perrotin, M. (2008). Towards compositional safety analysis via semantic representation of component failure behaviour. In *Proceedings of JCKBSE'08* (pp. 405–414).

Montani, S., Portinale, L., Bobbio, A., & Codetta-Raiteri, D. (2008). Radyban: A tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks. *Reliability Engineering & System Safety*, *93*(7), 922–932.

Niu, R., Tang, T., Lisagor, O., & McDermid, J. A. (2011). Automatic Safety Analysis of Networked Control System based on Failure Propagation Model. In *IEEE International Conference on Vehicular Electronics and Safety (ICVES)* (pp. 53–58).

Open AADL (2016). Ocarina AADL model processor. In: Available online at: http://www.openaadl.org/ocarina.html

Ortmeier, F., Reif, W., & Schellhorn, G. (2005). Deductive Cause-Consequence Analysis. In *Proceedings of the 6th IFAC World Congress* (pp. 1434–1439).

OSATE. (2016). OSATE. In: Available online at: http://osate.github.io/

Palshikar, G. K. (2002). Temporal fault trees. *Information and Software Technology*, *44*(3), 137–150.

Pande, P. K., Spector, M. E., & Chatterjee, P. (1975). *Computerized Fault Tree Analysis: TREEL and MICSUP*. California, USA.

Papadopoulos, Y. (2000). *Safety-Directed System Monitoring Using Safety Cases*. University of York, PhD Thesis.

Papadopoulos, Y. (2012). HiP-HOPS. *Dependable Systems Research Group, University of Hull*. Retrieved September 1, 2014, from http://hip-hops.eu/

Papadopoulos, Y., & Mcdermid, J. A. (1999). Hierarchically Performed Hazard Origin and Propagation Studies. In *Proceedings of the 18th International Conference on Computer Safety, Reliability and Security* (pp. 139–152).

Papadopoulos, Y., Walker, M., Parker, D., Rüde, E., Hamann, R., Uhlig, A., Grätz, U., & Lien, R. (2011). Engineering failure analysis and design optimisation with HiP-HOPS. *Engineering Failure Analysis*, *18*(2), 590–608. doi:10.1016/j.engfailanal.2010.09.025

Papadopoulos, Y., Walker, M., Parker, D., Sharvia, S., Bottaci, L., Kabir, S., Azevedo, L., & Sorokos, I. (2016). A synthesis of logic and bio-inspired techniques in the design of dependable systems. *Annual Reviews in Control*, *41*, 170–182. https://doi.org/10.1016/j.arcontrol.2016.04.008

Point, G., & Rauzy, A. (1999). AltaRica: Constraint automata as a description language. *European Journal on Automation*, *33*(8-9), 1033–1052.

PTC. (2016). Windchill Product Risk and Reliability. In: Available online at: http://www.ptc.com/product-lifecycle-management/windchill/product-risk-and-reliability

Pullum, L. L., & Dugan, J. B. (1996). Fault tree models for the analysis of complex computer-based systems. In *Proceedings of Annual Reliability and Maintainability Symposium* (pp. 200–207). doi:10.1109/RAMS.1996.500663

Rajakarunakaran, S., Kumar, A. M., & Prabhu, V. A. (2015). Applications of fuzzy faulty tree analysis and expert elicitation for evaluation of risks in LPG refuelling station. *Journal of Loss Prevention in the Process Industries*, *33*, 109–123. doi:10.1016/j.jlp.2014.11.016

Ramentor. (2016). ELMAS Fault Tree. In: Available online at: http://www.ramentor.com/products/elmas/fault-tree/

Rao, K. D., Gopika, V., Rao, V. V. S. S., Kushwaha, H. S., Verma, A. K., & Srividya, A. (2009). Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment. *Reliability Engineering & System Safety*, *94*(4), 872–883. doi:10.1016/j.ress.2008.09.007

Rauzy, A. (2002). Mode automata and their compilation into fault trees. *Reliability Engineering & System Safety*, *78*(1), 1–12.

ReliaSoft. (2016). BlockSim: System Reliability and Maintainability Analysis Software Tool. In: Available online at: http://www.reliasoft.com/BlockSim/index.html

RiskSpectrum. (2015). RiskSpectrum FTA. In: Available online at: http://www.riskspectrum.com/en/risk/Meny_2/RiskSpectrum_FTA/

Roth, M., & Liggesmeyer, P. (2013). Qualitative analysis of state/event fault trees for supporting the certification process of software-intensive systems. In *IEEE International Symposium on Software Reliability Engineering Workshops* (pp. 353–358).

Rugina, A., Kanoun, K., & Kaâniche, M. (2007). A System Dependability Modeling Framework Using AADL and GSPNs. In *Architecting Dependable Systems IV* (pp. 14–38).

Rugina, A., Kanoun, K., & Kaâniche, M. (2008). The ADAPT Tool : From AADL Architectural Models to Stochastic Petri Nets through Model Transformation. In *7th European Dependable Computing Conference (EDCC)* (pp. 85–90).

Ruijters, E., & Stoelinga, M. (2015). Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, *15*, 29–62. doi:10.1016/j.cosrev.2015.03.001

SAE (2012). *Architecture Analysis & Design Language (AADL) (AS5506B)*. Technical Report, SAE International, Warrendale, PA.

Semanderes, S. N. (1971). ELRAFT: A Computer Program for the Efficient Logic Reduction Analysis of Fault Trees. *IEEE Transactions on Nuclear Science*, *18*(1), 481–487.

Sharvia, S., Kabir, S., Walker, M., & Papadopoulos, Y. (2015). Model-based dependability analysis: State-of-the-art, challenges, and future outlook. In *Software Quality Assurance: In Large Scale and Complex Software-intensive Systems* (pp. 251–278). doi:10.1007/10367682_1

Shu, M.-H., Cheng, C.-H., & Chang, J.-R. (2006). Using intuitionistic fuzzy sets for fault-tree analysis on printed circuit board assembly. *Microelectronics Reliability*, *46*(12), 2139–2148. doi:10.1016/j.microrel.2006.01.007

Singer, D. (1990). A fuzzy set approach to fault tree and reliability analysis. *Fuzzy Sets and Systems*, *34*(2), 145–155.

Sokolsky, O., Lee, I., & Clark, D. (2006). Schedulability Analysis of AADL models. In *20th Parallel and Distributed Processing Symposium*.

Steiner, M., Keller, P., & Liggesmeyer, P. (2012). Modeling the effects of software on safety and reliability in complex embedded systems. In F. Ortmeier & P. Daniel (Eds.), *Computer Safety, Reliability, and Security* (Vol. 7613, pp. 454–465). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-33675-1

Sun, H., Hauptman, M., & Lutz, R. (2007). Integrating product-line fault tree analysis into AADL models. In *Proceedings of IEEE International Symposium on High Assurance Systems Engineering* (pp. 15–22). doi:10.1109/HASE.2007.46

Suresh, P. V., Babar, A. K., & Raj, V. V. (1996). Uncertainty in fault tree analysis : A fuzzy approach. *Fuzzy Sets and Systems*, *83*(2), 135–141.

Tanaka, H., Fan, L. T., Lai, F. S., & Toguchi, K. (1983). Fault-Tree Analysis by Fuzzy Probability. *IEEE Transactions on Reliability*, *R-32*(5), 453–457.

TDC Software. (2016). TDC FTA. In: Available online at: http://www.tdc.fr/en/products/tdc_fta.php

Tyagi, S. K., Pandey, D., & Kumar, V. (2011). Fuzzy Fault Tree Analysis for Fault Diagnosis of Cannula Fault in Power Transformer. *Applied Mathematics, 02*(11), 1346–1355.

US Department of Defense (1980). *Procedures for Performing a Failure Mode, Effects, and Criticality Analysis (MIL-STD-1629A)*. Washington DC, USA.

Verma, A. K., Srividya, A., Prabhudeva, S., & Vinod, G. (2006). Reliability analysis of Dynamic fault tree models using fuzzy sets. *Communications in Dependability and Quality Management*, *9*(4), 68–78.

Vesely, W. E., Goldberg, F. F., Roberts, N. H., & Haasl, D. F. (1981). Fault Tree Handbook. Washington DC, USA: US Nuclear Regulatory Commission.

Vesely, W. E., Stamatelatos, M., Dugan, J., Fragola, J., Minarick, J., & Railsback, J. (2002). Fault tree handbook with aerospace applications. Washington DC,USA: NASA Office of Safety and Mission Assurance.

Walker, M. (2009). *Pandora: A Logic for the Qualitative Analysis of Temporal Fault Trees*. University of Hull, PhD Thesis.

Walker, M., Bottaci, L., & Papadopoulos, Y. (2007). Compositional Temporal Fault Tree Analysis. In *Proceedings of the 26th International Conference on Computer Safety, Reliability and Security (SAFECOMP'07)* (pp. 106–119).

Walker, M., Mahmud, N., Papadopoulos, Y., Tagliabo, F., Torchiaro, S., Schierano, W., & Lonn, H. (2008). *ATESST2: Review of relevant Safety Analysis Techniques*.

Walker, M., & Papadopoulos, Y. (2009). Qualitative temporal analysis: Towards a full implementation of the Fault Tree Handbook. *Control Engineering Practice*, *17*(10), 1115–1125.

Wang, D., Zhang, P., & Chen, L. (2013). Fuzzy fault tree analysis for fire and explosion of crude oil tanks. *Journal of Loss Prevention in the Process Industries*, *26*(6), 1390–1398. doi:10.1016/j.jlp.2013.08.022

Wijayarathna, P. G., & Maekawa, M. (2000). Extending fault trees with an AND–THEN gate. In *Proceedings of the 11th International Symposium on Software Reliability Engineering* (pp. 283–292).

Xu, B., Huang, Z., Hu, J., Wei, O., & Zhou, Y. (2013). Minimal cut sequence generation for state/event fault trees. In *Proceedings of the 2013 Middleware Doctoral Symposium*.

Yang, L. P. (2011). Analysis on Dynamic Fault Tree Based on Fuzzy Set. *Applied Mechanics and Materials*, *110-116*, 2416–2420.

Yang, Y., Zeckzer, D., Liggesmeyer, P., & Hagen, H. (2011). ViSSaAn : Visual Support for Safety Analysis. *Dagstuhl Follow-Ups*, *2*, 378–395.

Yevkin, O. (2011). An improved modular approach for dynamic fault tree analysis. In *Proceedings of Annual Reliability and Maintainability Symposium* (pp. 1–5). doi:10.1109/RAMS.2011.5754437

Yuhua, D., & Datao, Y. (2005). Estimation of failure probability of oil and gas transmission pipelines by fuzzy fault tree analysis. *Journal of Loss Prevention in the Process Industries*, *18*(2), 83–88.

Zadeh, L. (1965). Fuzzy Sets. *Information and Control*, *8*(3), 338–353.

Zhang, P., & Chan, K. W. (2012). Reliability evaluation of phasor measurement unit using monte carlo dynamic fault tree method. *IEEE Transactions on Smart Grid*, *3*(3), 1235–1243.

Zhang, X., Miao, Q., Fan, X., & Wang, D. (2009). Dynamic fault tree analysis based on Petri nets. In *8th International Conference on Reliability, Maintainability and Safety(ICRMS)* (pp. 138–142). Chengdu: IEEE.