Model-based Dependability Analysis: State-of-the-art, Challenges, and Future Outlook

Abstract: Over the past two decades, the study of model-based dependability analysis has gathered significant research interest. Different approaches have been developed to automate and address various limitations of classical dependability techniques to contend with the increasing complexity and challenges of modern safety-critical system. Two leading paradigms have emerged, one which constructs predictive system failure models from component failure models compositionally using the topology of the system. The other utilizes design models - typically state automata - to explore system behaviour through fault injection. This paper reviews a number of prominent techniques under these two paradigms, and provides an insight into their working mechanism, applicability, strengths and challenges, as well as recent developments within these fields. We also discuss the emerging trends on integrated approaches and advanced analysis capabilities. Lastly, we outline the future outlook for model-based dependability analysis.

Keywords: Model-based Dependability Analysis, Safety Analysis, System Design, Failure Logic Modelling, Behavioural Fault Injection

Author Contact Information:

Septavera Sharvia

Postdoctoral Research Assistant, Computer Science

University of Hull

HU67RX Hull, UK

s.sharvia@hull.ac.uk

Telephone: +44 (0)1482 465253

Sohag Kabir

PhD Student, Computer Science

University of Hull

HU67RX Hull, UK

s.kabir@2012.hull.ac.uk

Telephone: +44 (0)1482 465951

Martin Walker

Lecturer, Computer Science

University of Hull

HU67RX Hull, UK

Martin.Walker@hull.ac.uk

Telephone: +44 (0)1482 465994

Yiannis Papadopoulos

Professor, Computer Science

University of Hull

HU67RX Hull, UK

Y.I.Papadopoulos@hull.ac.uk

Telephone: +44 (0)1482 465981

Model-based Dependability Analysis: State-of-the-art, Challenges, and Future Outlook

Information in This Chapter:

- Failure Logic Synthesis and Analysis
- Behavioural Fault Simulation
- Towards Integrated Approaches
- Challenges and Future Outlook

1 Introduction

Integrated and effective dependability analysis has become increasingly important as modern safety-critical systems become more heterogeneous and complex. Dependability can be defined as the "the ability of an entity to perform one or several required functions under given condition" (Villemeur, 1991). The study of system dependability covers four properties safety, reliability, availability, and maintainability. Safety is the ability of the system to avoid causing hazards for people and the environment. Reliability is the ability of the system to perform its intended functions satisfactorily for a given time interval. Availability studies the readiness of the system to perform its function at a given instance of time. And maintainability is the ability of the system to be maintained or restored to a state in which it can perform its function, when maintenance is performed as specified. In this paper, emphasis is placed on safety and reliability due to the context of safety-critical systems in which many of the techniques are situated. However, references are made to work within these techniques to address availability and maintainability, e.g. as in (Papadopoulos, Nggada, & Parker, 2010). The integration between analysis techniques and advanced system engineering and modelling is also beneficial for the functionality, accessibility, and usability dimensions of the system development.

Dependability assessment should begin early in the design phase so that potential problems can be identified and rectified early to avoid expensive changes in the later phase of the system lifecycle. Traditional dependability analysis techniques like fault tree analysis (FTA) (Vesely, Dugan, Fragola, Minarick, & Railsback, 2002) and Failure Modes and Effects Analysis (FMEA) (US Department of Defense, 1980) are well-established and widely used during the design phase of safety-critical systems. FTA is a deductive analysis technique which utilizes graphical representation based on Boolean logic to show logical connections between different failures and their causes. FMEA is an inductive technique which tries to infer the unknown effects on the system of known component failure modes.

These techniques are typically applied manually and often performed on an informal system model which may rapidly become out of date as the system design evolves. This presents challenges in maintaining the consistency and completeness of the assessment process.

Over the past 20 years, new developments in the field of dependability engineering have led to a body of work on model-based assessment and prediction of dependability. Model-based techniques offer significant advantages over traditional approaches as they utilise software automation and integration with design models to simplify the synthesis and analysis of complex safety-critical systems. These techniques can be applied from early stages of expressing requirements and until detailed architectural design.

Emerging model-based dependability analysis (MBDA) techniques can be conceptualized and classified according to different criteria. In (Aizpurua & Muxika, 2012), classification criteria include the type of traditional limitations overcome by new techniques, recovery strategies, and the approach to design verification. Classical FTA and FMEA are static in nature and do not take into consideration the time or sequence dependencies. They are also traditionally manual processes which rely heavily on the analysts' skills and are susceptible to human errors. Certain MBDA techniques have been developed to address the temporal and dynamic limitations, while other techniques focus on making the analysis process more manageable. Different techniques may also employ different recovery strategies, including heterogeneous redundancies, homogeneous redundancies, shared redundancies, graceful degradation, and implicit redundancies. Techniques have also been classified based on the type of design verification, i.e. whether it is based on fault injection or an integrative approach. In (Lisagor, Kelly, & Niu, 2011), MBDA are categorized based on the model provenance and the engineering semantics of the component interfaces. Model provenance categorizes techniques based on the construction of safety model and its relationship with the system design. Safety analysis models can be defined either through extension to the design model, or as a dedicated model defined by safety engineers. Regarding the engineering semantics of components, categorization is possible with respect to the type of modelling of component dependencies. These dependencies can be captured in terms of either deviations from design intent, or abstracted nominal flow (for example energy, matter and information).

The classification of MBDA techniques in this paper is based on the general underlying formalism and the types of analysis performed. Based on this, model-based techniques typically gravitate towards two leading paradigms. The first paradigm, termed Failure Logic Synthesis and Analysis (FLSA), focuses on the automatic construction of predictive system failure analyses, such as fault trees or FMEAs, on the basis of information stored in the system model. These approaches are typically compositional, where the system-level models of failure propagation can be generated from component-level failure logic and the overall topology of the system. This compositionality lends itself well to automation and reuse of component failure models across applications, and this is beneficial to dependability analysis in ways similar to those introduced by reuse of trusted software components in software engineering. Techniques which follow this approach include Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS), Component Fault Trees (CFT), State-Event Fault Trees (SEFT), and the Failure Propagation and Transformation Notation and Calculus (FPTN and FPTC).

The second paradigm, termed Behavioural Fault Simulation (BFS), automatically analyses potential failures in a system and the development has led to a group of formal verification based techniques. These generally work by injecting possible faults into simulations based on executable, formal specifications of a system and studying the effects of those faults on the system behaviour. The results are then used by model checking tools to verify whether system dependability requirements are being satisfied or whether violations of the requirements exist in normal or faulty conditions. Techniques in this category include AltaRica, The Formal Safety Analysis Platform/New Symbolic Model Verifier (FSAP-NuSMV), Safety Analysis Modelling Language (SAML), and Deductive Cause Consequence Analysis (DCCA).

Much of this recent work on dependability analysis has a natural synergy with a wider trend towards model-based design, particularly domain-specific languages. In many industries, particularly transport and aerospace, designers are increasingly adopting Architecture Description Languages (ADLs) to encapsulate both architectural and behavioural information about the system. Such ADLs may not only represent the system itself, but also the functional and non-functional requirements and properties of the system; they may also provide facilities for the refinement of the system throughout the design lifecycle, showing how the requirements are being met at each stage. One key aspect of such ADLs is to represent the safety requirements and the failure logic of the system, and these areas have often seen integration with model-based dependability analysis techniques. For instance, recent work has demonstrated that dependability analysis of automotive EAST-ADL models is possible via HiP-HOPS while dependability analysis of aerospace AADL error models is possible via conversion to classical artefacts e.g. combinatorial and temporal fault trees or Generalised Stochastic Petri Nets (GSPNs). The integration of the comprehensive behavioural and architectural data offered by ADLs with model-based analysis engines also makes new forms of analysis possible. This has subsequently led to techniques that allow automatic optimisation of system attributes — such as dependability, cost, and performance — by means of meta-heuristics that can efficiently explore the huge design spaces involved.

This work complements previous much less detailed discussions on classification and overview of model-based analysis techniques presented in (Lisagor, Kelly, & Niu, 2011) and

(Aizpurua & Muxika, 2013). We extensively explore various prominent techniques and study their recent updates and developments. The aim of this paper is not to define strict classification of techniques, but to review the state-of-the-art in this field, explaining the fundamental concepts involved and comparing the key techniques that have been developed in terms of their features, achievements, applicability, and scalability. We also discuss the current challenges faced by these techniques, including representativeness and completeness of models, modelling and analysis structure, the scalability of models and analyses, and obstacles in practicability and uptake of this work. We conclude with a discussion on the future outlook of this work, looking at how these challenges may be addressed and how research is already being developed to address new problems, including separation of hardware/software concerns in embedded systems, and efficient multi-objective optimisation of different system attributes.

The remainder of this paper is structured as the following: Section 2 discusses a number of prominent FLSA techniques. Section 3 discusses a number of techniques employing Behavioural Fault Simulation. As techniques mature, further development tends to blur the lines of categorization, and techniques often extend into a hybrid or integrated approach. Section 4 studies a number of emerging integrated techniques and challenges, while Section 5 concludes and outlines future outlook.

2 Failure Logic Synthesis and Analysis

In FLSA, system failure models are constructed from component failure models using a process of composition. System failure models typically comprise, or can be automatically converted into, well-known dependability evaluation models such as fault trees, stochastic Petri-nets and Markov chains. These types of techniques therefore model the deviation from the design intent rather than nominal (normal) behaviour of the system.

Techniques based on FLSA include the Failure Propagation and Transformation Notation (FPTN), Failure Propagation and Transformation Calculus (FPTC), Hazard Origin and Propagation Studies (HiP-HOPS), Component Fault Trees (CFT), and State-Event Fault Trees (SEFT). Architecture Description Languages (ADLs) are widely adopted in the recent years to support the integration between analysis models in FLSA and system design models expressed in the language. An introduction to an ADL called Architecture Analysis and Design Language (AADL) is therefore included in this section.

2.1 Failure Propagation and Transformation Notation (FPTN)

Failure Propagation and Transformation Notation (FPTN) (Fenelon & McDermid, 1993) is among the pioneering MBDA methods designed to address limitations of FTA and FMEA in specifying system failure behaviour. It was developed as part of the Software Safety Assessment Procedures (SSAP) project. It uses a modular, hierarchical notation to describe the propagation of faults through the modules of system architecture. FPTN module is represented as a box with a set of inputs and outputs, which can be connected to other modules. To form a hierarchical structure, each module can contain a number of sub-modules. Failures can be propagated or transformed from one type to another. The relation between the inputs and outputs is expressed by a set of logical equations equivalent to the minimal cutsets (smallest and necessary combination of failures which cause a higher-level fault) of the fault trees describing the output failure modes of the module. Therefore, each module represents a number of fault trees describing all the failure modes for that module. These equations can also contain more advanced constructs, allowing FPTN to represent recovery mechanisms and internal failure modes. This type of notation enables FPTN to be used both inductively (to create an FMECA) and deductively (to create an FTA).

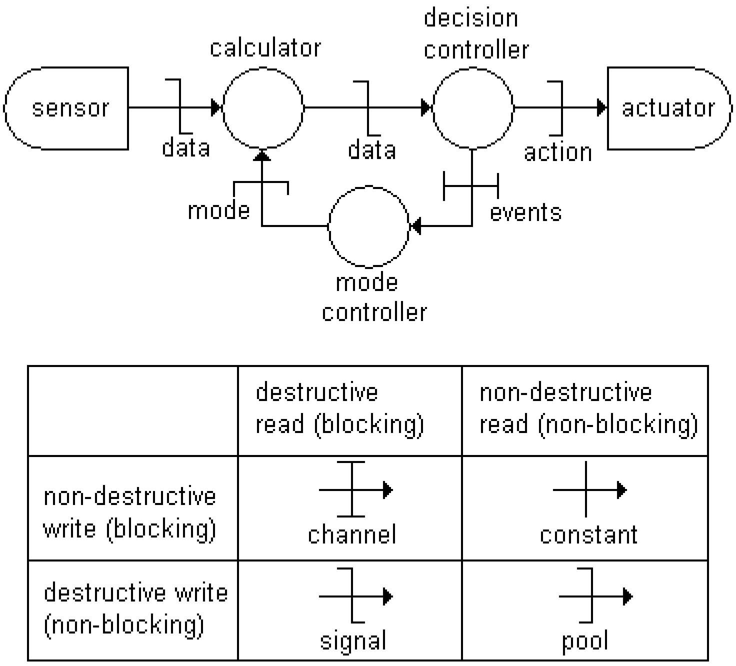
FPTN is designed to be developed alongside the design of the system. Information collected can then be used to identify potential flaws and problems in the system design so that they can be eliminated or compensated for in the next design iteration. In its classical form, FPTN is limited to static analysis, but recent work on Temporal-FPTN (Niu, Tang, Lisagor, & McDermid, 2011) extended FPTN with temporal information to allow dynamic analysis by using Failure Temporal Logic to specify failure relationship, and produces Minimal Cutset Sequences. However, although FPTN provides a systematic and formal notation for representing the failure behaviour of a system (a distinct improvement on traditional ad hoc approaches), it lacks full automation, and the fact that each analysis must be conducted manually hampers the opportunity for it to be used in an iterative design process.

2.2 Failure Propagation & Transformation Calculus

Failure Propagation & Transformation Calculus (FPTC) (Paige, Rose, Ge, Kolovos, & Brooke, 2008) is a method for the representation and analysis of the failure behaviour of the software and hardware components of a system. It allows annotation of an architectural model of a system with concise expressions describing how each component can fail; these annotations can then be used to compute the failure properties of the whole system automatically.

FPTC is primarily designed for real-time software systems where a statistically schedulable code unit is considered as the primary unit of the architectural description. The data and control flow behaviour of the system is defined by connecting these units using communications protocols like handshakes and buffers. FPTC assumes that all the threads and communications are known in advance and are not created or destroyed dynamically during the system operation. FPTC also offers the capability to describe the allocation of these units and their communications to different physical processing devices and networks. This makes it possible to describe how, for example, one faulty processor can affect all software units running on it.

FPTC represents the system architecture using a RTN (Real-Time Network) style notation, consisting of a graph of arcs and nodes representing hardware and software units and the communications between them. Communications are typed according to protocol (e.g. blocking / non-blocking). RTN offers significant capabilities, including the ability to refine graphs hierarchically, to define code units as state machines, and to automatically generate Ada code from the design. An example RTN graph and associated key to some communications protocols are illustrated in Figure 1.



	destructive read (blocking)	non-destructive read (non-blocking)
non-destructive write (blocking)	channel	constant
destructive write (non-blocking)	signal	pool

Figure 1 Example of FPTC Representation in RTN (Walker *et al*., 2008)

Similar to FPTN, components may respond to failures in one of two ways: by propagating the failure or by transforming the failure. Components may also initiate or terminate failures, e.g. by failing silent or by detecting and correcting failures. These failures are typed similarly to FPTN e.g. timing, value, omission failures, but the types are not fixed and can be extended as required. 'Normal' is also a type, indicating the lack of a failure.

The reaction to failure is described by a simple pattern-based notation. Once components are annotated with FPTC expressions, the resulting RTN graph can then be thought of as a tokenpassing network in which failure tokens flow from one node to another, being created, transformed, and destroyed along the way. Each arc in the graph can then be further annotated with the set of possible failures that may propagate along it. This is done by 'running' each expression in reaction to the 'normal' type and listing the resulting output failures on the output communication arcs; each component is then re-run in response to any new input failure types. The process terminates when no more new output failures are generated and all possible input/output combinations have been considered.

One advantage of FPTC is the fact that it uses the full architectural models used for developing the software code and can adapt much more readily to changes . This helps ensure the FPTC model is synchronised with the design and offers significant advantage compared to FPTN which is annotated according to *known* failures, and so any new failure to be added requires the whole model to be manually reannotated. It also handles cycles in architectures by using fix-point calculations. There is also support for model transformation from AADL and SysML models through Epsilon (Paige, Rose, Ge, Kolovos, & Brooke, 2008). Recently, a probabilistic type known as Failure Propagation and Transformation Analysis (FPTA) is proposed in (Ge, Paige, & McDermid, 2009). This method links architectural models with probabilistic model checkers specified in PRISM and allows FPTC to capture non-deterministic failure behaviour. FI^{4}FA (Gallina & Punnekkat, 2014)is the most recent extension of FPTC which allows FPTC to consider more types of failure behaviour e.g. incompletion, inconsistency, interference and impermanence and also analysis of mitigating behaviours. The primary disadvantage of FPTC is the necessity of performing a different analysis for each failure or combination of failures to be considered. Each originating failure must be specified at a component and then the model must be re-analysed to determine how this failure will propagate through the system and what failure modes will be communicated to critical components.

2.3 Component Fault Tree

Component Fault Trees (CFTs) (Kaiser, Liggesmeyer, & Mackel, 2003) is an extension to traditional fault trees which aims to provide better association between the hierarchy of faults and the architectural hierarchy of the system components. Although traditional fault tree allows modularization, it provides little information on the hierarchical decomposition of the physical system. CFTs define smaller fault trees for each component, thus incorporating the fault trees as part of the hierarchical component model of the system. Like traditional fault trees, CFTs use basic events, logical gates as well as input and output ports. The fact that CFTs are still logical structures linking output failures to input causes means that they can be analysed qualitatively and quantitatively using standard fault tree algorithms.

CFTs differ from traditional fault trees in the sense that they allow multiple top events and by representing repeating (or common cause) failures only once. CFTs also form directed acyclic graphs called Cause Effect Graphs (CEGs), as illustrated in Figure 2, instead of traditional tree structure. The use of CEGs makes the CFTs smaller and easier to analyse, both significant benefits when modelling large systems. It also makes the diagrams clearer, as the fault tree nodes can be displayed as part of their components.

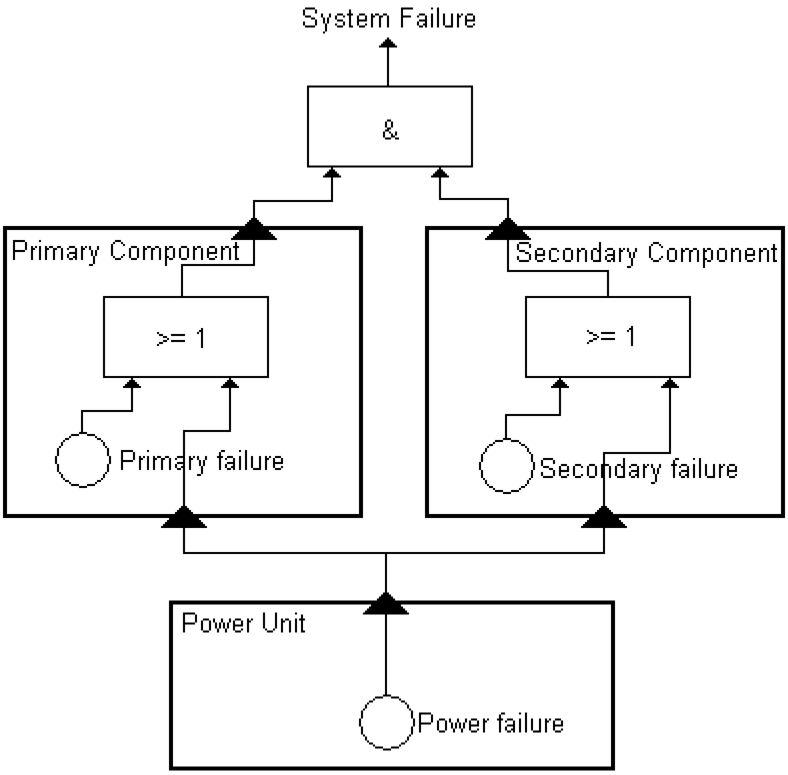


Figure 2 Example of Component Fault Tree (Kaiser, Liggesmeyer, & Mackel, 2003)

The main advantage of CFTs is its capability of hierarchical decomposition of systems to manage the complexity of modern systems. CFTs create smaller fault trees for each of the components and neatly capture the hierarchical system architecture. Consequently, different parts of the system can be developed and stored separately as part of the component definition in a library, and this facilitates greater degree of reusability. Conceptually, this hierarchical decomposition also makes it possible for the failure behaviour of the system to be modelled at different levels, e.g. for the top level subsystems first, and then once the design has been refined further, for the sub components as well.

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 efficient representation of information related to minimal cut sets of CFTs. (Adler, et al., 2011) have developed a metamodel to extract reusable CFTs from the functional architecture of systems specified in UML.

2.4 State-Event Fault Tree

One of the limitation of FTA is its inability to adequately account for the temporal order of events, whether in terms of a simple sequence or a set of states and transitions. This limits the capability to analyse complex systems, particularly real-time embedded and software based systems. Fault trees are fundamentally a combinatorial technique and are not well suited to modelling the dynamic behaviour in such systems. State-Event Fault Trees (SEFTs) (Grunske, Kaiser, & Papadopoulos, 2005) are developed to address this limitation by combining elements from fault trees with Statecharts and Markov chains. This is done by adding states and events to fault trees, allowing the use of system state-based models as the basis for the analysis, as well as enabling the use of more sophisticated analysis methods (e.g. Markov chains). SEFTs can also be seen as an evolution of CFTs in that they allow decomposition and distribution across the components of the system, and represent inputs and outputs as visible ports in the model.

SEFTs make a distinction between causal transition and sequential relation, and therefore provide corresponding separate types of ports. A sequential transition applies to states which specify a predecessor or successor relation between states, whereas a causal transition applies to events which define a causal (trigger/guard) relationship between events. Because events are explicitly represented (and do not always have to be state transitions), it is also possible for one event to cause another event. These events can also be combined using traditional fault tree gates (e.g. AND and OR) so that a combination of events is necessary to trigger another event. SEFTs also offer more advanced features for modelling timing scenarios. For example, events can be assigned deterministic or probabilistic delays by means of Delay Gates and SEFTs also allow the use of NOT gates. Sequential and causal modelling is further refined by means of History-AND and Priority-AND gates, which can check whether an event has occurred in the past and in what order it occurred, and other gates are also possible, e.g. Duration gates to ensure that a state has been active for a given amount of time.

In SEFTs, 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 state transition. Each component has its own state space and each component can only be in one state at any point in time (the 'active state'). For the purposes of quantitative analysis, probabilities can be assigned to each state to reflect its chance of being the active state at any time. Similarly, events may be assigned probability densities for quantitative analysis.

SEFTs follow the same general procedure as standard FTA in the modelling of system failure behaviour. Analysts begin with the occurrence of a system failure and trace it back through the components of the system to determine its root causes. SEFTs offer a greater level of detail during this analysis , e.g. by considering the effect of states in fault propagation. SEFTs also allow a greater degree of reuse than traditional fault trees because pre-existing state charts from the design can be used, as can Markov chains, which can be similarly integrated into the SEFTs.

Unlike CFTs, SEFTs can no longer be analysed using traditional FTA algorithms. The inclusion of states and the different modelling of events means that different techniques are needed, such as the conversion to Petri Nets, to allow for the calculation of probabilities of system failures. Steiner, Keller, & Liggesmeyer (2012) have proposed a methodology to create and analyse SEFTs based on the ESSaRel tool (ESSaRel, 2005). SEFT models are converted to Deterministic Stochastic Petri Nets (DSPNs) (Marsan & Chiola, 1987), then the analysis of the DSPN models can be performed using a DSPN analyser like TimeNET (German & Mitzlaff, 1995). The conversion process requires the consideration of the entire system, which can lead to an explosion of state-spaces and thus performance problems for larger system models. This issue can be alleviated to some degree by using both combinatorial FTA-style algorithms and dynamic state-based algorithms to analyse different parts of the system, e.g. using the faster techniques for simple, static subsystems and using slower but more powerful techniques for the dynamic parts of the system. The effectiveness of this dual-analysis technique will depend heavily on the type of system being analysed.

2.5 Hierarchically Performed Hazard Origin & Propagation Studies

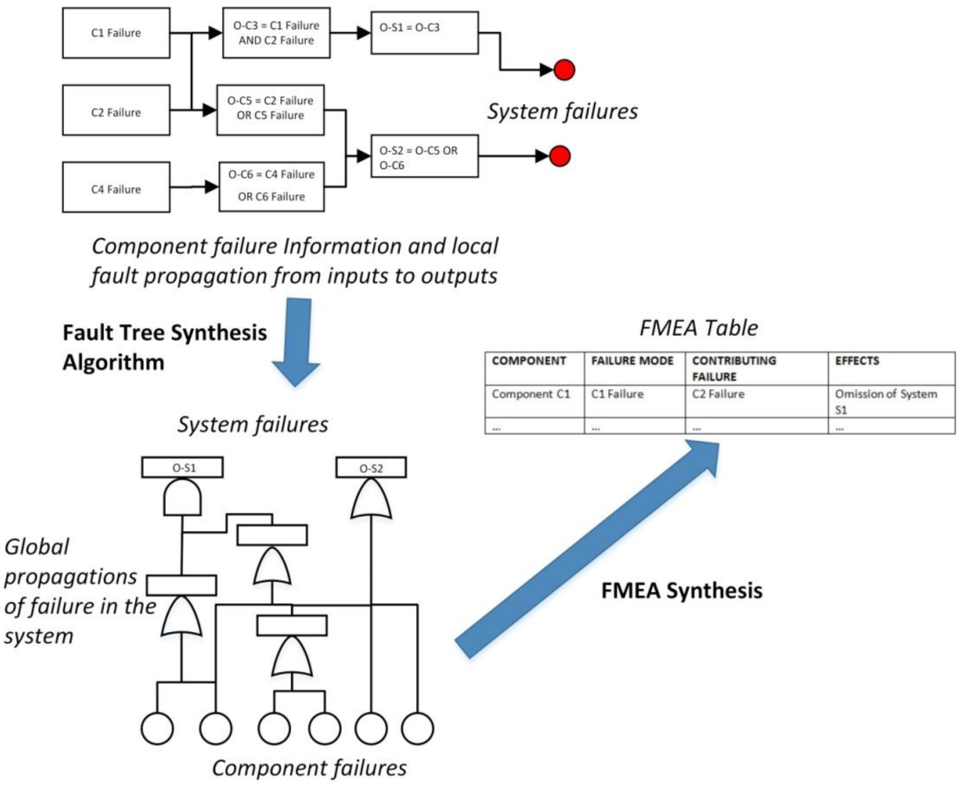
Hierarchically Performed Hazard Origin & Propagation Studies (HiP-HOPS) (Papadopoulos & McDermid, 1999) is one of the pioneering MBDA techniques, and amongst the well-supported advanced compositional safety analysis techniques. It provides a greater degree of automation compared to CFT or FPTN. HiP-HOPS also supports automatic optimisation of designs (Adachi, Papadopoulos, Sharvia, Parker, & Tohdo, 2011) (Papadopoulos, et al., 2011) which can be employed for selection among alternatives for components and subsystems as well for optimal decisions on the level and location of replicated components. Recently HiP-HOPS has also been extended with capabilities for top-down automatic allocation of safety requirements in the form of Safety Integrity Levels (SIL). The latter automates some of the processes for Automotive SIL (ASIL) allocation as specified in the safety standard ISO26262.

HiP-HOPS works in conjunction with commonly-used system modeling tools, for example Matlab Simulink or Simulation X. Failure editors are integrated into these modelling tools to allow system designers to annotate components with failure information. This failure information includes failure modes (internal malfunction) and output failure expressions, and describes how the component fails and its relationship with other component failures, i.e. whether and how the component responds or not to effects of failure received at the component inputs. HiP-HOPS takes this information and examines how the component failures propagate through the system topology, producing sets of interrelated fault trees and eventually an FMEA. This approach also enables the hierarchical structure of the system to be captured neatly in the fault trees. There are three main phases in HiP-HOPS: model annotation, fault tree synthesis, and fault tree and FMEA analysis phase.

The model annotation phase provides information to HiP-HOPS on how the component can fail. It takes the form of a set of expressions which are manually added. These local failure expressions describe how failures of the component outputs can be caused by a combination of failures received at the component's inputs and/or by internal malfunctions of the component itself. Common cause failures are also supported, as are failures propagated via other means, e.g. from allocated components. In this way it is possible to model more sophisticated scenarios — for instance, the effects on a software function, and consequently the software architecture, when the processor shown in the hardware architecture to be executing that function fails.

The synthesis phase produces an interconnected network of fault trees which link system-level failures (i.e., failures of the system's output functions) to component-level internal failures by using the model topology and component failure information. These fault trees show how component failures propagate from one component to another and how ultimately they may affect the wider system, whether individually or in combination with other component failures.

In the analysis phase, the synthesized fault trees are analyzed via automated algorithms to generate minimal cut sets. Minimal cut sets describe the necessary and sufficient combination of events which lead to the undesired events. Eventually the data is combined into a multiple failure mode FMEA which shows both direct effects of failure modes on the system, as well as the further effects of the failure modes caused in conjunction with other failure modes occurring in the system. The resultant FMEA is presented in tables which are conveniently displayed through a web browser. These main phases of HiP-HOPS are illustrated in Figure 3.


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
...	...	...	...

Figure 3 Main Phases in HiP-HOPS

Quantitative data can also be entered for the component to estimate the probability of internal failures occurring and the severity of output deviations. This data can then be used in the quantitative analysis phase to calculate the unavailability, i.e. failure probability, of the top event. HiP-HOPS assists reusability by enabling failure-annotated components to be stored in a library. This allows other components of a similar type to reuse failure data, and avoids the designer having to enter the same failure data multiple times. Recently, HiP-HOPS has also been extended with advanced features, including the capability to accommodate temporal analysis and perform multi-objective optimization.

2.5.1 Temporal Analysis in HiP-HOPS using Pandora

HiP-HOPS fundamentally inherits the static nature of FTA and this includes the lack of capability to capture time information or sequence-dependent behaviour. While the compositional failure model may be sufficient to describe the systems behaviour in many scenarios, it may not be adequate to describe the complete behaviour of complex systems. This drawback is particularly limiting in a system where functions and failure modes change according to different states. In addition to this, the ability to understand and capture the order of failures can be important in producing an accurate failure model. Pandora (Walker, Bottaci, & Papadopoulos, 2007) was proposed to extend traditional fault trees with dynamic analysis capability by introducing new temporal gates and temporal logic. This technique can be used to obtain minimal cut sequences (the smallest necessary sequences of events to cause the top events) of temporal fault trees. Pandora is based around the redefinition of the long-established Priority-AND (PAND) (Fussel, Aber, & Rahl, 1976) gate and aims to solve the ambiguities in the original PAND gate whilst maintaining the simplicity and flexibility of FTA. It allows the temporal ordering of events to be represented as part of the fault tree structure, and uses temporal logic gates Priority-AND (PAND), Priority-OR (POR), and Simultaneous-AND (SAND) to represent the temporal relations.

Pandora provides better modelling for precise failure behaviour of dynamic systems than ordinary fault tree analysis. Because Pandora is designed to integrate with existing Boolean logic, it can be used in existing tools such as HiP-HOPS, extending it with additional dynamic FTA capabilities. The solution proposed by (Merle, Roussel, Lesage, & Bobbio, 2010) is used as an analytical solution to Pandora's PAND gate. Quantitative analysis of Pandora is recently discussed in (Edifor, Walker, & Gordon, 2012) and (Edifor, Walker, Gordon, & Papadopoulos, 2014)

2.6 Architecture Analysis and Design Language

The Architecture Analysis and Design Language (AADL) (Feiler & Rugina, 2007) is a domain-specific language developed for the specification and analysis of hardware and software architectures of performance-critical real-time systems. AADL enables an array of modelling capabilities including structural description of the system as an assembly of software components mapped onto an execution platform, functional description of interfaces to components, and performance description of critical aspects of components. AADL allows both architectural modelling and error modelling. Architectural modelling describes the nominal architecture of the system, including the components, and their connections and interactions. Interactions show structural and behavioural aspects without considering the presence of faults. In contrast, error modelling captures the behaviour of components in the presence of internal faults, repair events, as well as external propagations of faults from other components.

An AADL error model consists of a model type and, at least, one error model implementation. The form of error models is described in the AADL error model annex, which was intended to support the qualitative and quantitative analysis of dependability attributes. The error model is a state machine that can be associated with an AADL element, i.e. component or connection, in order to describe its behaviour in terms of logical error states in the presence of faults. The error model can be associated with software (e.g. process, data, thread), hardware (e.g. processor, memory, device) and composite component (e.g. system) component and connection (Feiler, Gluch, & Hudak, 2006). In AADL, systems may be represented as collections of components, hierarchies of components, or systems of systems. Therefore an AADL error model extends from system to subsystem to component, and the system error model is a composition of the error models of its subsystems or components. This captures hazards at system level, risk mitigation architecture at subsystem level and failure modes and effects analysis models at component level.

Each AADL error model can be stored in a library and can be reused for different AADL components. Propagation of errors between components is determined by their interdependencies and the AADL Error Model Annex has defined a set of dependency rules

(Feiler and Rugina, 2007) to define interdependencies between components. For example, propagations may occur from a component to any outgoing connections and between all sub-components of the same process which is conceptually similar to Papadopoulos's dual approach to propagations in his integration of HiP-HOPS and Matlab Simulink models (Papadopoulos & Maruhn, 2001).

One limitation of the language lies in the incomplete support, at least in its core concepts, of analysis of the runtime architectures. This is compensated by extensions to accommodate analysis specific notations that can be associated with components. Error modelling for instance is supported through an annex that has been added to the standard. AADL error models can be analyzed through an automated translation into a standard fault tree (Joshi, Vestal, & Binns, 2007), or by generating Generalized Stochastic Petri Nets (GSPNs) from error model specifications and using a GSPN tool for quantitative analysis (Rugina, Kanoun, & Kaaniche, 2007).

2.7 System Dynamics and Temporal Considerations

A number of techniques have been developed to address the temporal and dynamic limitation of classical FTA and FMEA. The Dynamic Fault Tree (DFT) (Dugan, Bavuso, & Boyd, 1992) approach introduced new gates and temporal notions to account for ordered events and handle probabilistic timed behaviour in fault trees. Some techniques which are based on Monte Carlo Simulation (MCS) (Rao *et al*., 2009) offer alternatives by modelling temporal failure and repair through state-time diagrams. Dynamic Reliability Block Diagram (DRBD) (Distefano & Puliafito, 2007) model component failures and repairs based on their dependencies using state machines, while coloured Petri Nets have been used to analyzed DRBD in (Robidoux, Lu, Xing, & Zhou, 2010). Other researchers like (Hura & Atwood, 1988), and Helmer et al. (2007) have used Petri Nets to solve classical fault trees. The combination of state and event based formalisms has been adopted in Boolean Logic Driven

Markov Processes (Bouissou, 2007) and State-Event Fault Trees (SEFT). The temporal extension to HiP-HOPS which is implemented through Pandora also aims to address temporal dynamics.

3 Behavioural Fault Simulation

In Behavioural Fault Simulation, system failure models are produced by injecting faults into executable formal specifications of a system, thereby establishing the system level effects of faults. This fault injection technique was developed in the ESACS (Bozzano *et al*, 2003) and ISAAC (Akerlund *et al.* 2006) projects. As opposed to the dedicated analysis model commonly used in FLSA, Behavioural Fault Simulation uses an extended model which is automatically constructed from a system design model. The extended model typically contains both the nominal input flow as well an input related to the fault, which is taken into consideration when activated. System behaviour is observed when faults are activated. The fundamental analysis of this approach is similar to the exhaustive search through activation of all possible combinations of failures.

Model checking is often used to verify the system safety properties in the extended model. Model checking performs exhaustive exploration to check whether a safety property – which is usually expressed in temporal logic – holds. The tool produces counterexample when safety properties do not hold to show traces of simulation on how the breaching condition is reached.

3.1 Formal Safety Analysis Platform - New Symbolic Model Verifier

The Formal Safety Analysis Platform - New Symbolic Model Verifier (FSAP/NuSMV-SA) (Bozzano & Villafiorita, 2003) 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 and allows failure injection into the system. The effects of that failure on the system behaviour are then observed.

The FSAP/NuSMV-SA platform has different modules to perform different tasks. The central module of the platform is the SAT Manager which control the other modules of the platform. It stores all the information related to safety assessment and verification which includes the extended system model, failure modes, safety requirements and analyses. System models are described as finite state machines using the NuSMV language as plain text. A 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 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 the model checking capability incorporated in the NuSMV-SA Model Checker module. This module also generates counter examples and safety analysis results by means of fault trees. The Result Extraction and Displayer 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 textual or graphical or tabular fashion.

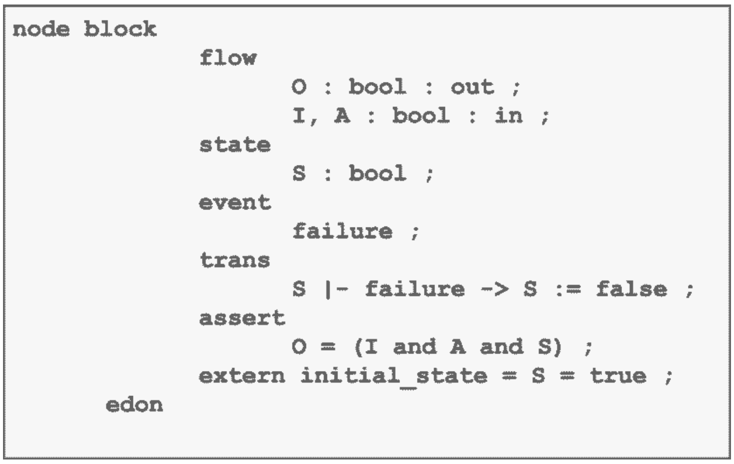
Although the FSAP/NuSMV-SA platform provides multiple functionalities; it does also have some limitations, especially in handling fault trees. Fault trees generated by this toolset show the relation between top events and basic events. However fault trees are flat and don't show propagation of failure which could make the fault trees for complex systems unintuitive. The tool enables qualitative FTA, but it does not have the ability to perform probabilistic evaluation of FTs. Like other model checking based approaches, this platform also suffers from state space explosion while modelling large or complex systems.

3.2 AltaRica

AltaRica (Point & Rauzy, 1999) is a description language designed to be able to formally describe complex systems. AltarRica allows systems to be represented as hierarchies of components and subcomponents and models both events and states. Unlike FSAP/NuSMV, AltaRica uses dedicated safety models. AltaRica models can be analysed by external tools and methods, e.g. for the generation and analysis of fault trees, Petri nets, and model-checking (Bieber, Castel, & Seguin, 2002).

In AltaRica, components are represented as nodes, and each node possesses a number of states and variables (either state variables or flow variables). The values of the state variables are local to the node they are in, and change when an event occur, i.e. events are triggering state transitions, thus changing the values of state variables. Flow variables are visible both locally and globally and are used to provide an interface to other nodes in the model.

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 small example of AltaRica node is shown in Figure 4.



```
node block
            flow
                  O : bool : out ;
                  I, A : bool : in ;
            state
                  S : bool ;
            event
                  failure ;
            trans
                  S |- failure -> S := false ;
            assert
                  O = (I and A and S) ;
            extern initial_state = S = true ;
      edon
```

Figure 4 Small Example of AltaRica Node

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 (event) and a guard that constraints the transition; guards are assertions over flow and state variables. In the example, the node 'block' contains three flow variables (O, I, A) and one state variable (S). There is one event, failure, that causes the state to transition to false. The assertion links the flow variables such that output only occurs when input is present, an active signal is present, and the component is functioning (i.e. S = true, which is the initial state).

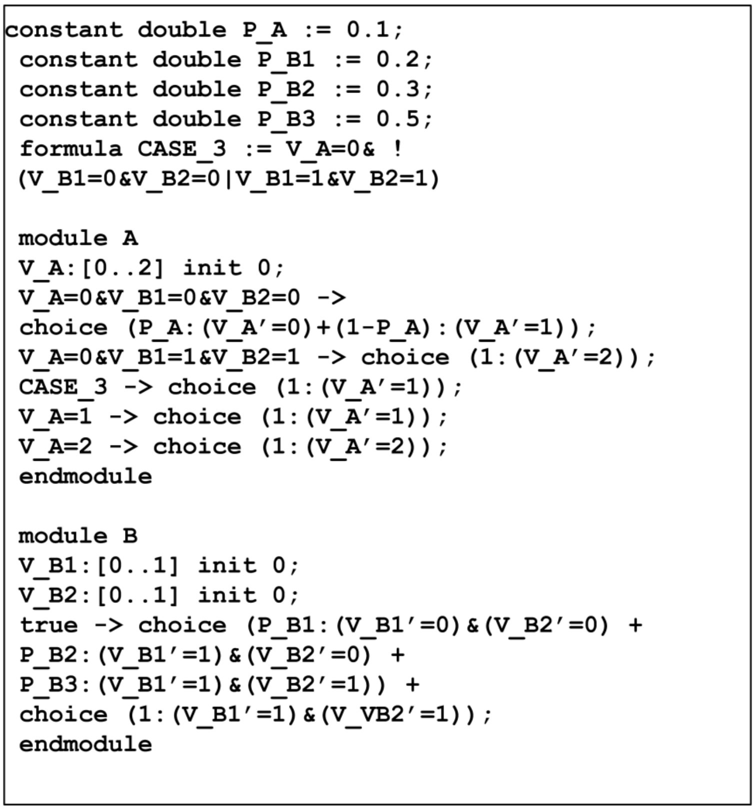
After defining the nodes, these can be organized hierarchically to reflect system decomposition and architecture. The top-level node represents the system itself, and it consists of other lower-level nodes. Nodes can communicate either through interfaces or though event dependencies. The first process 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 node. 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.

Two main variants of AltaRica have been designed so far. The primary difference between the variants is how the variables are updated after firing of transitions. In the original AltaRica (Arnold, Point, Griffault, & Rauzy, 2000) , variables are updated by solving constraints, and thus consume too much computational resource. Therefore, this approach is not scalable for industrial application although it is very powerful. To make AltaRica capable of assessing industrial scale applications, AltaRica Data-Flow (Boiteau, Dutuit, Rauzy, & Signoret, 2006) is 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, however, it cannot naturally model bidirectional flows through a network, cannot capture located synchronisation, and faces difficulties in modelling looped systems. Recent work on AltaRica 3.0 (Batteux, Prosvirnova, Rauzy, & Kloul, 2013) is under specification. It improves the expressive power of the second version without reducing the efficiency of assessment algorithms. The main improvement is: it defines the system model as a new formalism - that of Guarded Transition Systems (GTS) - which allows modelling systems with loops, and can easily model bidirectional flows. AltaRica 3.0 provides a set of assessment tools, e.g. a Fault Tree generator, a Markov chain generator, and stochastic and stepwise simulators.

3.3 Safety Analysis Modelling Language

The Safety Analysis Modelling Language (SAML) (Güdemann & Ortmeier, 2010) is a tool-independent modelling framework that can be used to 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 system models consisting of hardware, software components and environment inputs. 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.

A SAML model consists of at least one module description and declarations of zero or more constants and formulas. Figure 5 shows an example of SAML model.



```
constant double P_A := 0.1;
 constant double P_B1 := 0.2;
 constant double P_B2 := 0.3;
 constant double P_B3 := 0.5;
 formula CASE_3 := V_A=0& !
 (V_B1=0&V_B2=0|V_B1=1&V_B2=1)

 module A
 V_A:[0..2] init 0;
 V_A=0&V_B1=0&V_B2=0 ->
 choice (P_A:(V_A′=0)+(1-P_A):(V_A′=1));
 V_A=0&V_B1=1&V_B2=1 -> choice (1:(V_A′=2));
 CASE_3 -> choice (1:(V_A′=1));
 V_A=1 -> choice (1:(V_A′=1));
 V_A=2 -> choice (1:(V_A′=2));
 endmodule

 module B
 V_B1:[0..1] init 0;
 V_B2:[0..1] init 0;
 true -> choice (P_B1:(V_B1′=0)&(V_B2′=0) +
 P_B2:(V_B1′=1)&(V_B2′=0) +
 P_B3:(V_B1′=1)&(V_B2′=1)) +
 choice (1:(V_B1′=1)&(V_VB2′=1));
 endmodule
```

Figure 5 Example of SAML Model

This example has two modules (A and B), four constants, and one formula. The modules are declared as state automata, so every module has at least one state variable and at least one rule to update the state variable. Every state variable is represented as a range of integer values and initialised with a value. Every update rule defines either at least a probabilistic assignment or at least one non-deterministic choice of assignments, and they are conditioned on a Boolean activation condition.

SAML models can be transformed automatically to the input format of other model-based safety analysis techniques. 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 different targeted tools.

Güdemann & Ortmeier (2011) have shown ways of transforming SAML model into the input language of probabilistic model checker PRISM (Kwiatkowska, Norman, & Parker, 2011). In the same work, the above researchers have also shown ways of transforming SAML modules to NuSMV although the former supports both non-deterministic and probabilistic update rules whereas the later one supports only non-deterministic update rules.

In addition to being a high-level modelling and specification language, SAML can also be used as an intermediate language. Its formal qualitative and quantitative semantics allows different analyses to be performed in the same model.

3.4 Deductive Cause Consequence Analysis

Deductive Cause Consequence Analysis or DCCA (Ortmeier, Reif, & Schellhorn, 2005) is a formal method for safety analysis which uses mathematical methods to determine whether a given component fault is the cause of a system failure. It is a formal generalization of FMEA and FTA, but it is 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 system failure. This property is known as *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).

DCCA also faces state explosion problem because to determine minimal critical sets it has to consider all possible combinations of component failure modes. This problem can be alleviated to some extent by using results from other informal safety analysis techniques like FTA which are believed to be generating smaller but good initial guesses for solutions. However, by doing this, DCCA also inherits the shortcomings of FTA, i.e. inability of capturing dynamic behaviour where order of events is important. Deductive Failure Order Analysis (Güdemann, Ortmeier, & Reif, 2008) is a recent extension which enables DCCA to deduce temporal ordering information in critical sets. 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.

4 Towards Integrated Approaches

This section explores the strengths and limitations often shared by different techniques within the FLSA and BFS fields. There has been a paradigm shift in recent years where research work and efforts have been channelled into extending and integrating different techniques to address identified limitations.

4.1 Applicability and Challenges of FLSA

FLSA techniques generally use a dedicated model developed for the purpose of the analysis (or annotations that augment the design model), which makes it easier to analyse the effect of failures on the system. This allows safety engineers to modify level of details avoiding unnecessary complexity while ensuring that the model is sufficient for dependability analysis purposes. Unintentional interactions (e.g. short circuits of electrical systems) can also be taken into consideration.

The true benefits of this type of approach are most apparent when used as part of an iterative design process. As the failure behaviour of the system components is modelled in a compositional fashion, it is easier to determine the effects of design changes. This is particularly true for automated or partly automated techniques, which speed up the analysis process and make it possible to rapidly evaluate speculative changes to the design. This efficient nature of FLSA also means that valuable analysis can be started early in the design process when concrete system details are still scarce. FLSA produces safety artefacts which are familiar to safety engineers (e.g. FTA and FMEA).

However, dedicated models also mean that additional effort is required to create these new models or extend any normal system model with the required information, and further effort may be required to harmonise these disparate models. This may also hamper the traceability between design and analysis models.

Another limitation of FLSA is the lack of support for formal verification. FLSA are also fundamentally static analyses, which do not take into consideration the changes in system states and are therefore limited in their ability to capture dynamic behaviour (although this limitation is, to a certain extent, addressed by some extended techniques as previously mentioned).

4.2 Applicability and Challenges of BFS

The strength of BFS lies in its ability to facilitate automated formal verification and capture the system dynamic behaviours. It is also possible to distinguish between transient and permanent failures and model the temporal ordering of failures. However, the fault simulation techniques have a number of limitations. The valuable safety artefacts such as fault trees (which are obtained through model checking) tend to have a 'flat structure', representing disjunction of all minimal cut-sets. This may hamper the understanding of the fault trees. Model-checking based techniques are computationally expensive, inductive in nature, and therefore suffer from state-space explosion problems. The exhaustive assessment of the effects of combinations of component failures is not feasible in large systems.

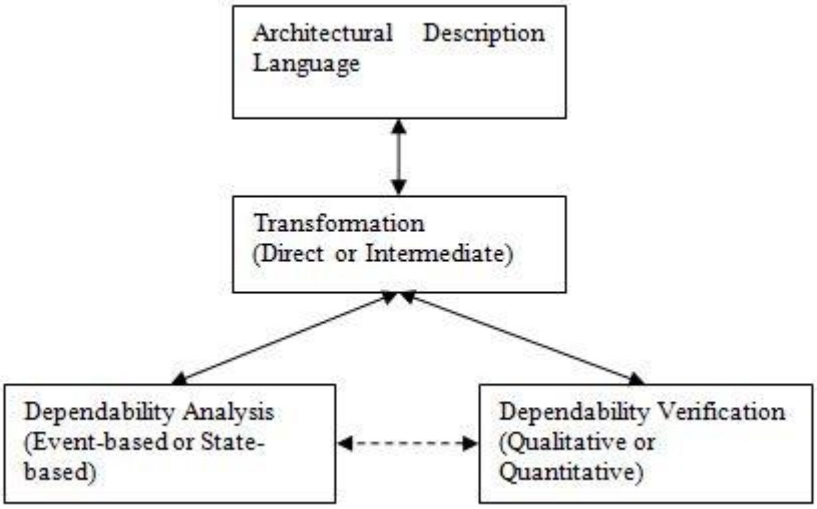
Fault injection is also typically applied to executable design models, which are typically produced at a later development process stage when design changes are costly to implement.

The analysis results therefore often lose the opportunity to drive the design process itself. While the construction of the extended model supports the consistency of the safety analysis, it may impose constraints on the safety analysis as explained in (Lisagor, Kelly, & Niu, 2011). Extended models are inadequate in covering failures resulting from unintentional interactions or unintended dependencies between seemingly unrelated components. The techniques also rely on the set of predefined failure modes to be injected, and therefore the completeness of the analysis depends on the completeness of the failure list, which is difficult to guarantee.

4.3 Towards Integrated Approaches

As MBDA techniques develop and mature, various extensions are introduced to address the limitations identified. One of the increasing trends in integrated approaches is that between Architecture Description Languages (ADLs) and FLSA techniques. FLSA techniques aim to overcome the problems associated with a 'pure' dedicated model by automatically or semi-automatically constructing the dependability analysis model (by partially utilizing the architecture of the design model). Translations from high-level ADLs to FLSA techniques allow tighter integration between the design and analysis process, and therefore a better traceability between design and analysis models. Recent work on FPTC in (Paige, Rose, Ge, Kolovos, & Brooke, 2008) uses a metamodel to support model transformations from SysML and AADL models. Metamodels have also been developed in (Adler, et al., 2011) to obtain CFT models from architectural models specified in UML. HiP-HOPS has been integrated with Matlab Simulink and Simulation X for many years (Papadopoulos & Maruhn, 2001). Recent integration work between HiP-HOPS and EAST-ADL is discussed in (Chen, Mahmud, Walker, Feng, Lonn, & Papadopoulos, 2013) and (Sharvia S. , Papadopoulos, Walker, Chen, & Lonn, 2014), and model transformation between HiP-HOPS and AADL is presented in (Mian, Bottaci, Papadopoulos, Sharvia, & Mahmud, 2014).

Another trend on integration aims to enable the verification capabilities in FLSA. A number of integrated approaches have emerged where compositional FLSA techniques are merged with fault injection approaches. In this integrated approach, system structure and behaviour (nominal and failure) is expressed using a compositional model of architecture, and model transformation is performed to obtain a model which can be formally verified (verification model). The transformation can be carried out through direct transformation rules, or through an intermediate transformation where the intermediate model is used to achieve consistency and traceability between different design, analysis and verification approaches. The outline of this structure is illustrated in Figure 6. With the use of the intermediate transformation, high level models can be reused for different target approaches. Counter examples which are obtained from fault injection techniques can also be transformed back into dependability analysis models, for example as minimal cut-sets of fault trees. Failure Propagation and Transformation Analysis (FPTA) (Ge, Paige, & McDermid, 2009) links FPTC architectural models with probabilistic model checkers specified in PRISM. This enables FPTC to capture non-deterministic failure behaviour, and perform verification. Combined application between HiP-HOPS and NuSMV is described in (Sharvia & Papadopoulos, 2011). This work describes how dependability analysis results from HiP-HOPS can be used to systematically guide the construction of verification models specified in NuSMV, allowing early verification of functional behaviour and formulation of system degradation states.



Architectural Description Language
Transformation (Direct or Intermediate)
Dependability Analysis (Event-based or State-based)
Dependability Verification (Qualitative or Quantitative)

Figure 6 Model-based Integrated Approach (Aizpurua & Muxika, 2013)

Integration through direct transformation is used in COMPASS project (COMPASS, 2013) which utilizes the System Level Integrated Modelling (SLIM) language (Bozzano, Cimatti, Katoen, Nguyen, & Noll, 2011). SLIM semantics covers nominal and error behaviour of AADL, and contains an extended model which allows verification via model checking. SAML is an example of an integrative approach via intermediate transformation. This is illustrated in Figure 7. A specification can be written in high-level CASE tool like SCADE or Matlab Simulink, transformed into a SAML model, and verified using NuSMV or PRISM. However, work on this is still in progress and there are transformation-related issues which need to be addressed (for example, implementation of high level to intermediate level model transformation). Another example is the Topcased project (TOPCASED, 2013) which transforms SysML, UML and AADL models into an intermediate model specified in FIACRE language (Berthomieu, et al., 2008).

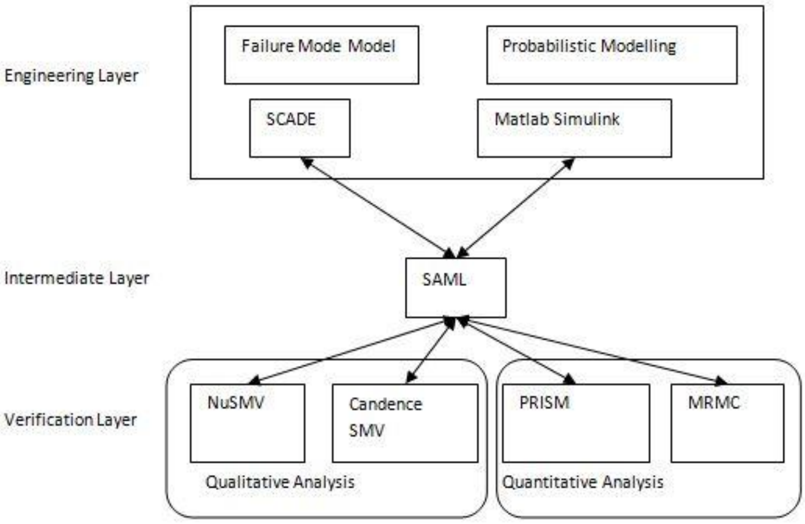


Figure 7 SAML as an Intermediate Language (Güdemann, Lipaczewski, Struck, & Ortmeier, 2012)

Challenges in these integrated approaches include the state-space explosion problem due to the size of verification model. There is also need to find an efficient way to feed the analysis results into the design model in order to maintain consistency, and a need to construct a user friendly toolset due to the range of models and analyses.

The following table summarizes the approaches discussed in this paper:

Technique	**Features**	**Limitations**	**Tool Support**	**Extension/ Integrated Work**
FPTN	Formal notation for system behavior; Temporal extension	Lack of full automation	SSAP toolset	N/A
FPTC	Integration with probabilistic model checker	Manual identification of all potential failure modes of interest	Epsilon	AADL, SysML, PRISM
CFT	Associations between fault and architectural hierarchies	Lack of dynamic behavioral analysis	ESSaRel tool	UML

SEFT	Capture system dynamic behavior	Conversion to state-based models for analysis; State-explosion	ESSaRel, TimeNET	N/A
HiP-HOPS	Automated dependability analysis; Temporal extension; Multi-objective optimization	Lack of mature dynamic behavioral analysis	HiP-HOPS tool	EAST-ADL, AADL, Simulink, Simulation X, NuSMV
FSAP/NuSMV	Formal verification; Library of failure modes	Flat fault tree structure; State-explosion	FSAP/NuSMV	N/A
AltaRica	Formal verification; Timed automata and GTS extension	State-explosion	AltaRica tools	AADL
SAML	Captures deterministic and probabilistic behavior	Manual extension of nominal model	S^3 tools	NuSMV, ADL (ongoing)
SLIM	Integrated formal verification and dependability analysis	Manual extension of nominal model; State-explosion	ESA toolset (Internal)	ESA toolset (Internal)

4.4 Conclusions and Future Outlook

Various MBDA techniques have been developed over the past 20 years, and these techniques tend to gravitates towards two different paradigms. This paper discussed the characteristics of both paradigms, and reviewed a number of prominent techniques, exploring their working mechanism, strengths, limitations and recent developments. These techniques have also evolved with recent extensions and integrations (as discussed in section 4.3) and utilize different strengths to address various challenges outlined earlier. In line with the increasing adoption of ADLs which encapsulate both architectural and behavioural information of the system, recent work has seen a number of model transformations between pioneering MBDA techniques and ADL models to enable greater analysis capabilities and consistency between design and analysis. This addresses the challenges arising from the use of dedicated model and improves the traceability between design and analysis models. Other types of integration aim to extend the analysis capabilities of the MBDA technique itself, particularly to enable verification in conjunction with dependability analyses. With the increasing popularity of model-driven engineering, metamodels for techniques have also been constructed to assist automation of code generations and model transformations.

Future trends are likely to yield more robust integrations between existing paradigms and techniques. Efforts should also be placed into exploring ways to utilize different strengths in a complementary manner. The dependability community will also benefit from integrated automated tools to support adoptions of various techniques with minimum overhead caused by disjoint and dysfunctional tool chains. Separation concerns for hardware and software within design of complex embedded systems have, to a certain extent, been supported through the integration of analysis techniques with ADLs. Concerns still exist about traceability between models and analysis and focus should be given to feeding analyses effectively back to the design. 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).

Advanced capabilities to support the development and design decision of safety-critical systems are also important, particularly in a modern competitive engineering environment. The design of dependable systems must often address both cost and dependability concerns. The availability of different component alternatives and architectural configurations means that the task to find optimal or near optimal solutions is not a trivial one. It is also possible that no architectural configuration is able to meet all design requirements. In this case, the optimal trade-offs between dependability and cost need to be established. This opens the field to multi-objective optimization. MBDA techniques like HiP-HOPS have been extended with multi-objective optimization capabilities to assist design decisions (Adachi, Papadopoulos, Sharvia, Parker, & Tohdo, 2011); and Eclipse-based tool, ArcheOpterix, allows evaluation techniques and optimization heuristics for AADL specifications (Aleti, Bjornander, Grunske, & Meedeniya, 2009). Other works which look into the use of reconfigurable architectures for fault tolerant design and recovery strategies are discussed in (Aizpurua & Muxika, 2013) and (Papadopoulos, et al., 2011). There is also opportunity for model-based allocation of dependability requirements to be used as a tool for driving design refinement itself. This topic is for example studied in recent works within HiP-HOPS (Azevedo, Parker, Walker, Papadopoulos, & Araujo, 2013) where the automated allocation of safety requirements in the form of Safety Integrity Levels is investigated.

Bibliography

Adachi, M., Papadopoulos, Y., Sharvia, S., Parker, D., & Tohdo, T. (2011). An Approach to Optimisation of Fault Tolerant Architecture using HiP-HOPS. *Software: Practice and Experience* , 1202-1327.

Adler, R., Domis, D., Hofig, K., Kemmann, S., Kuhn, T., Schwinn, J., et al. (2011). Integration of Component Fault Trees into the UML . *Workshops and Symposia at MODELS.*, (pp. 312 - 327).

Aizpurua, J. I., & Muxika, E. (2012). Design of Dependable Systems: An Overview of Analysis and Verification Approaches. *DEPEND 2012: Fifth International Conference on Dependability* (pp. 4-12). IARIA.

Aizpurua, J., & Muxika, E. (2013). Model-based Design of Dependable Systems: Limitation and Evolution of Analysis and Verification Approaches. *International Journal on Advances in Safety* .

Akerlund, O., & Bieber, P. (2006). ISAAC, a Framework for Integrated Safety Analysis of Functional, Geometrical, and Human Aspecs. *3rd European Congress on Embedded Real Time System (ERTS)* . Toulouse, France.

al., W. e. (2008). *ATESST2: Review of Relevant Safety Analysis Techniques.*

Aleti, A., Bjornander, S., Grunske, L., & Meedeniya, I. (2009). ArcheOpterix: An Extendable Tool for Architecture Optimization of AADL Models. *MOMPES'09.* Vancouver, Canada.

Arnold, A., Point, G., Griffault, A., & Rauzy, A. (2000). The AltarRica Formalism for Describing Concurrent System. *Fundamenta Informaticae* , 109-124.

Azevedo, L., Parker, D., Walker, M., Papadopoulos, Y., & Araujo, R. (2013). Assisted Assignment of Automotive Safety Requirements. *IEEE Software* .

Batteux, M., Prosvirnova, T., Rauzy, A., & Kloul, L. (2013). The AltaRica 3.0 Project for Model-based Safety Assessment. *INDIN* .

Berthomieu, B., Bodeveix, B., Farail, M., Garavel, H., Gaufillet, P., Lang, F., et al. (2008). Fiarce: an Intermediate Language for Model Verification in Topcased environment. *ERTS'08.*

Berthomieu, B., Ribet, P., & Vernadat, F. (2004). The Tool TINA - Construction of Abstract State Spaces for Petri Nets and Time Petri Nets. *International Journal of Production Research* .

Bieber, P., Castel, C., & Seguin, C. (2002). Combination of Fault Tree Analysis and Model Checking for Safety Assessment of Complex System. *Proceedings of the 4th European Depting Conference on Dependable Computing* , (pp. 19-31).

Boiteau, M., Dutuit, Y., Rauzy, A., & Signoret, J. (2006). The AltarRica Dataflow Language in use: Modelling of Production Availability of a Multi-State System. *Reliability Engineering and System Safety* , 747-755.

Bouissou, M. (2007). A Generalization of Dynamic Fault Trees through Boolean Logic Driven Markov Processes (BDMP). *Proceedings of ESREL '07* , 1051-1058.

Bozzano, M., & Villafiorita, A. (2003). Improving System Reliability via Model Checking: The FSAP/NuSMV-SA Safety Analysis Platform. *International Conference on Computer Safety, Reliability, and Security.* Edinburgh.

Bozzano, M., Cimatti, A., Katoen, J., Nguyen, V., & Noll, T. (2011). Safety, Dependability, and Performance Analysis of Extended AADL Models. *Computer Journal* , 754-775.

Bozzano, M., Villafiorita, A., & etal. (2003). ESACS: an Integrated Methodology for Deisgn and Safety Analysis of Complex Systems. *ESREL '03.*

Chen, D.-J., Mahmud, N., Walker, M., Feng, L., Lonn, H., & Papadopoulos, Y. (2013). Systems Modelling with EAST-ADL for Fault Tree Analysis through HiP-HOPS. *4th IFAC Workshop on Dependable Control of Discrete Systems.*

COMPASS. (2013). *Correctness, Modelling, and Performance of Aerospace Systems*. Retrieved from www.compass.informatik.rwth-aachen.de

Distefano, S., & Puliafito, A. (2007). Dynamic Reliability Block Diagramvs Dynamic Fault Trees. *Proceedings of Reliability Availability Maintainability Safety '07*, (pp. 71-76).

Dugan, J., Bavuso, S., & Boyd, M. (1992). Dynamic Fault Tree Models for Fault Tolerant Computer Systems. *IEEE Transactions on Reliability* , 363-377.

Edifor, E., Walker, M., & Gordon, N. (2012). Quantification of Priority-OR Gates in Temporal Fault Trees. *Computer Safety, Reliability, and Security SE* , 99-110.

Edifor, E., Walker, M., Gordon, N., & Papadopoulos, Y. (2014). Using Simulation to Evaluate Dynamic Systems with Weibull or Lognormal Distributions. *Proceedings of the 9th International Conference on Dependability and Complex Systems* , 177-187.

ESSaRel. (2005). *Embedded Systems Safety and Reliability Analyser*. Retrieved September 3, 2014, from http://essarel.de.

Feiler, P., & Rugina, A. (2007). *Dependability Modeling with the Architecture Analysis & Design Language (AADL).* Carnegie Mellon University.

Feiler, P., Gluch, D., & Hudak, J. (2006). *The Architecture Analysis & Design Language (AADL) : an Introduction.*

Fenelon, P., & McDermid, J. (1993). An Integrated Toolset for Software Safety Analysis. *Journal of Systems and Software* , 279-290.

Fussel, J., Aber, E., & Rahl, R. (1976). On the Quantitative Analysis of Priority-AND Failure Logic. *IEEE Transactions on Reliability* .

Gallina, B., & Punnekkat, S. (2014). A Formalism for Incompletion, Inconsistency, Interference and Impermanence Failures' Analysis. *Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications,* (pp. 493-500).

Ge, X., Paige, R., & McDermid, J. (2009). Probabilistic Failure Propagation and Transformation Analysis. *International Conference on Computer Safety, Reliability, and Security (SAFECOM)* , (pp. 215-228).

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

Grunske, L., Kaiser, B., & Papadopoulos, Y. (2005). Model-driven Safety Evaluation with State-Event-Based Component Failure Annotations. *Component Based Software Engineering 8th International Symposium* , (pp. 33-48).

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

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

Güdemann, M., Lipaczewski, M., Struck, S., & Ortmeier, F. (2012). Unifying Probabilistic and Traditional Formal Model Based Analysis. *MBEES 2012.*

Güdemann, M., Ortmeier, F., & Reif, W. (2008). Computation of Ordered Minimal Critical Sets. *Proceedings of the 7th Symposium in Computer Safety, Reliability, and Security* .

Hura, G., & Atwood, J. (1988). The Use of Petri Nets to Analyze Coherent Fault Trees. *IEEE Transactions on Reliability* , 469-474.

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

Kaiser, B., Liggesmeyer, P., & Mackel, O. (2003). A New Component Concept for Fault Trees. *Proceedings for the 8th Australian Workshop on Safety Critical Systems and Software (SCS '03)* .

Kwiatkowska, M., Norman, G., & Parker, D. (2011). PRISM 4.0: Verification of Probabilistic Real-time Systems. *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11),* (pp. 585-591).

Lisagor, O., Kelly, T., & Niu, R. (2011). Model-Based Safety Assessment: Review of Discipline and its Challenges.

Marsan, M., & Chiola, G. (1987). On Petri Nets with Deterministic and Exponentially Distributed Firing Times. *Advances in Petri Nets*, (pp. 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* , 250-261.

Mian, Z., Bottaci, L., Papadopoulos, Y., Sharvia, S., & Mahmud, N. (2014). Model Transformation for Multi-objective Architecture Optimization of Dependable Systems. *Dependability Problems of Complex Information Systems* , 91-110.

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

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

Paige, R., Rose, L., Ge, X., Kolovos, D., & Brooke, P. J. (2008). FPTC: Automated Safety Analysis for Domain Specific Languages. *Proceedings of Workshop on Non Functional System Properties in Domain Specific Modelling Languages*, (pp. 229-242).

Papadopoulos, Y., & Maruhn, M. (2001). Model-based Synthesis of Fault Trees from Matlab Simulink Models. *International Conference on Dependable Systems and Networks (DSN)* , 77-82.

Papadopoulos, Y., & McDermid, J. (1999). Hierarchically Performed Hazard Origin and Propagation Studies . *International Conference on Computer Safety, Reliability and Security*, (pp. 139-152).

Papadopoulos, Y., Walker, M., Parker, D., Rude, E., Hamman, R., Uhlig, A., et al. (2011). Engineering Failure Analysis & Design Optimization with HiP-HOPS. *Journal of Engineering Failure Analysis* .

Point, G., & Rauzy, A. (1999). AltaRica: Constraint Automata as a Description Language. *European Journal on Automation* , 1033-1052.

Rao, K., Durga, V., Gopika, V., Sanyasi, R., Kushawa, H., Verma, A., et al. (2009). Dynamic Fault Tree Analysis using Monte Carlo Simulation in Probabilistic Safety Assessment. *Reliability Engineering and System Safety* , 872-883.

Robidoux, R., Lu, H., Xing, L., & Zhou, M. (2010). Automated Modelling of Dynamic Reliability Block Diagrams using Coloured Petri Nets. *IEEE Transactions on Systems, Man, and Cybernatics* , 337-351.

Rugina, A., Kanoun, K., & Kaaniche, M. (2007). A System Dependability Modelling Framework using AADL and GSPNs. *Architecting Dependable Systems IV* , 14-38.

Sharvia, S., & Papadopoulos, Y. (2011). IACoB-SA: an Approach towards Integrated Safety Assessment. *Proceedings of 7th IEEE International Conference on Automation Science and Engineering.* Trieste, Italy.

Sharvia, S., Papadopoulos, Y., Walker, M., Chen, D., & Lonn, H. (2014). Enhancing the EAST-ADL Error Model with HiP-HOPS Semantics. *Athens ATINER Conference Paper Series.*

Steiner, M., Keller, P., & Liggesmeyer, P. (2012). Modelling the Effects of Software on Safety and Reliability in Complex Embedded Systems . *Computer Safety, Reliability, and Security*, (pp. 454-465).

TOPCASED. (2013). *The Open Source Toolkit for Critical System*. Retrieved November 2014, 09, from www.topcased.org

US Department of Defense. (1980). *Procedures of Performing a Failure mode, Effects, and Criticality Analysis* . Washington DC.

Vesely, W., Dugan, J., Fragola, J., Minarick, & Railsback, J. (2002). *Fault Tree Handbook with Aerospace Applications.*

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

Walker, M., Bottaci, L., & Papadopoulos, Y. (2007). Compositional Temporal Fault Tree Analysis. *Proceedings of the 26th International Conference on Computer Safety*, (pp. 106-119).

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