A Synthesis of Logic and Bio-inspired techniques in the Design of Dependable Systems

Yiannis Papadopoulos*, Martin Walker, David Parker, Septavera Sharvia, Leonardo Bottaci, Sohag Kabir, Luis Azevedo, Ioannis Sorokos

University of Hull, Hull, HU6 7RX, UK
*(*corresponding author: +44 (0)1482 465981, e-mail: Y.I.Papadopoulos@hull.ac.uk).*

Abstract: The technologies of model-based design and dependability analysis in the design of dependable systems, including software intensive systems, have advanced in recent years. Much of this development can be attributed to the application of advances in formal logic and its application to fault forecasting and verification of systems. In parallel, work on bio-inspired technologies has shown potential for the evolutionary design of engineering systems via automated exploration of potentially large design spaces. We have not yet seen the emergence of a design paradigm that effectively combines these two techniques, schematically founded on the two pillars of formal logic and biology, throughout the design lifecycle. Such a design paradigm would apply these techniques synergistically and systematically from the early stages of design to enable optimal refinement of new designs which can be driven effectively by dependability requirements. The paper sketches such a model-centric paradigm for the design of dependable systems that brings these technologies together to realise their combined potential benefits.

Keywords: dependability, safety integrity levels, genetic algorithms, MBSA, HiP-HOPS

1. INTRODUCTION

Dependability is an umbrella term that covers safety, reliability, availability, maintainability and security. Integrated and effective dependability assessment has become increasingly important as modern safety-critical systems become more heterogeneous and complex. Dependability assessment should begin early in the design so that potential problems can be identified and rectified early to avoid expensive changes later in the system lifecycle. Traditional dependability analysis techniques like fault tree analysis (FTA) and Failure Modes and Effects Analysis (FMEA) are well-established and widely used during the design phase of safety-critical systems. However, these techniques are manual processes and often performed on informal system models which may rapidly become out of date as the system design evolves.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 which has come to be known as Model-Based Safety Assessment (MBSA). MBSA focuses on safety but extends to other attributes of dependability including reliability, availability, and even assessment of implications of security on safety. Model-based techniques offer significant advantages over traditional approaches as they utilise software automation and integration with design models to simplify the analysis of complex safety-critical systems.

The various MBSA techniques generally fall into two leading paradigms. The first focuses on the automatic construction of predictive system failure analyses, such as fault trees or FMEAs, from local failure logic stored in the architectural model of the system, or a parallel error model. This approach is 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 logic 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 are based upon this paradigm include the Failure Propagation and Transformation Notation (Fenelon and McDermid, 1993) and Calculus (Wallace, 2005), Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) (Papadopoulos and McDermid, 1999), Component Fault Trees (Kaiser *et al.*, 2003) and State-Event Fault Trees (Grunske *et al.*, 2005).

The second prominent MBSA paradigm focuses on automatically analysing potential failures in a system model, typically represented as a state machine, using formal verification techniques such as model-checking. This generally works by injecting possible faults into an executable formal specification of a system and studying the effects of 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 (Arnold *et al.*, 2000), FSAP-NuSMV (Bozzano and Villafiorita, 2007), SAML (Ortmeier *et al.*, 2012) and PRISM (Kwiatkowska *et al*, 2009).

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 capture architectural and behavioural information about the system. Such ADLs may not only represent the architecture of the system, but also its functional and non-functional requirements; 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 important goal of such ADLs is to represent safety requirements and the failure logic of the system, and this has naturally led to integration with MBSA techniques.

Some of this work has been transferred to the context of model-based design. For instance, ADLs have incorporated error modelling semantics that enable dependability analysis. Recent work has demonstrated that dependability analysis of EAST-ADL models (Chen *et al.*, 2011) and SysML models (Andrews *et al.*, 2013) is possible via HiP-HOPS while dependability analysis of AADL models is possible via conversion to combinatorial (Joshi *et al.*, 2007) and temporal/ dynamic fault trees (Mahmud *et al.*, 2012; Merle *et al.*, 2014) or Generalised Stochastic Petri Nets (GSPN) (Feiler and Rugina, 2007).

This work is very much ongoing and there are specific challenges to be addressed within individual techniques and the field as a whole.

In this paper, we firstly discuss a set of challenges that in our view cannot be addressed by MBSA in its current state. These challenges mainly refer to design problems where there are many potential design options to be considered. Secondly, we argue that a synthesis of these techniques with modern metaheuristics for search and optimisation can potentially address these challenges. Finally, we describe our work towards this goal within the HiP-HOPS method and tool, and we show how this work can support cost-optimal, dependability-directed design refinement and optimisation of system architectures.

The paper is structured as follows: in Section 2, we discuss challenges; in Section 3, we present an extension of MBSA with metaheuristics; in Section 4, we discuss some of the technical challenges and limitations of the work; in Section 5, we discuss related work elsewhere in the literature, and in Section 6 we conclude by discussing how this work could inform the evolution of MBSA.

2. CHALLENGES

MBSA techniques can answer important questions regarding the quality of individual design proposals, and in that sense they can enrich a model-driven development process. However, MBSA is neither a panacea in its various forms nor is it a static field of research. Rather it is a set of techniques which are continuously evolving to address current and new challenges. Below we identify four such challenges which MBSA techniques cannot fully address at present.

2.1. Controlling dependability from the early stages

There is increasing agreement that to achieve high dependability in complex systems, design processes should move in a direction where dependability and other quality attributes are controlled from the early stages rather than left to emerge (or not) at the end. This is clearly a very desirable goal that would greatly benefit several industries, and it is enshrined in contemporary standards like the aerospace and the automotive safety standards, ARP4754-A and ISO 26262. These documents prescribe processes in which dependability requirements, captured early through system level hazard analysis and risk assessment, are rationally allocated to progressively more refined subsystem elements of the architecture in the form of Development Assurance Levels, Safety Integrity Levels, or other similar concepts.

A study of the problem (Parker *et al.*, 2013) has shown that the manual processes described in the standards become complex when applied to large or even medium-sized networked architectures which deliver multiple functions; such systems lead to huge numbers of potential allocation solutions and exploring these manually is infeasible. Current standards do not advise on how this type of allocation can be done effectively, for example with the support of automated algorithms and tools. This is an area where research opportunities arise to address important questions: for instance, which architectural proposals will fulfil dependability requirements better in the context of design refinement, and, given a proposed architecture, how integrity requirements can be optimally allocated to its elements.

2.2 Controlled design refinement over a complex value chain

The controlled refinement of a design for new dependable systems must be achieved in the context of a value chain over which the design and procurement of subsystems and components is typically distributed. Indeed, in practice, complex systems are developed in value chains using a combination of existing and commissioned subsystems and components that become parts of the overall architecture. Distributing a design in such a way that properties are verifiably maintained is a significant challenge that encompasses two aspects. On the one hand, effective top-down mechanisms are needed to ensure that the allocation and transmission of requirements during refinement is done in a way that satisfies overall requirements at the end. On the other hand, bottom-up mechanisms are needed to provide evidence that requirements have been met when the system is finally put together in its final form.

2.3 Dealing with the inevitable design changes

There is always a degree of uncertainty in early design, which often contributes to disruptive design changes. Some of the uncertainty comes as existing requirements are modified and new requirements are added, causing changes in the current design and allocation of requirements. These changes need to propagate in a top-down manner through the design refinement as new and modified requirements for the elements of the design. For example, a previously undiscovered hazard may cause design changes aimed at addressing the hazard and these may need to propagate through subsystems to the low levels of design and to the low level tiers of the value chain. Uncertainties in design may also propagate bottom up, for example in cases where assumptions about certain properties of elements cannot be satisfied. The challenge here is in being able to respond effectively and efficiently to the changes that need to follow. The impact of such changes should be localised and any necessary reallocation of requirements must be done efficiently and with minimum disruption to the contractual relationships between the various stakeholders in the tiers of the value chain.

2.4 Trading off dependability versus cost & other properties.

In complex distributed systems, rich functionalities and their distribution across shared hardware and communication channels allow a large number of configuration options at design time and a large number of reconfiguration options at runtime. This creates challenges in design because, as potential design spaces expand, their exploration for suitable or optimal designs becomes increasingly difficult.

When a number of different architectural configurations can potentially deliver the functions of a system, designers are faced with a difficult optimisation problem. Assuming that it is technically and economically possible to fulfil all dependability requirements, they must find an architecture that entails minimal development and other lifecycle costs. On the other hand, if fulfilling or optimising all dependability and other requirements is infeasible, then they must find the architecture or architectures that achieve the best possible tradeoffs among quality attributes and cost. The problem is compounded by the fact that attributes are often conflicting, e.g. improving safety often means not only increasing costs but also reducing availability.

It is widely accepted that these trade-offs represent hard, multi-objective optimisation problems that require optimisation algorithms that can search in large design spaces.

Although many design problems can be tackled effectively only by the human intellect, as design spaces expand, finding optimal designs in terms of quality and cost becomes increasingly difficult and some automation is needed. Modelling languages, emerging ADLs, and MBSA techniques could therefore benefit from concepts and technological support that enable this type of optimisation.

3. SYNTHESIS OF MBSA WITH METAHEURISTICS

The above challenges go beyond the capabilities of current MBSA techniques. We believe one step towards addressing them is to achieve a synthesis of MBSA and contemporary metaheuristics, i.e., moving into an area where formal logic can meet biology and nature-inspired techniques.

In recent years, we have been working in this direction in the context of HiP-HOPS, an MBSA technique which has been developed since the late 90s (Papadopoulos and McDermid, 1999). While HiP-HOPS started as a technique for model-based synthesis of fault trees and FMEAs, it has progressively evolved into a more sophisticated method in which heuristics are used to address the design problems highlighted in section 2 — see for instance (Papadopoulos and Grante, 2003 & 2006), (Zeng *et al.*, 2007), (Adachi *et al.*, 2011), (Papadopoulos *et al.*, 2011) (Walker *et al.*, 2013) and (Azevedo *et al.*, 2014).

The approach has been implemented as a prototype tool of the same name with a broad spectrum of capabilities for dependability analysis, architectural optimisation, and safety requirement allocation. Versions are in use by industrial partners including Volvo, Honda, Toyota, Embraer, Honeywell, and others.

3.1 Scope of HiP-HOPS

HiP-HOPS aspires to support both sides of the V engineering lifecycle with techniques that are model-based and automated (Fig. 1).

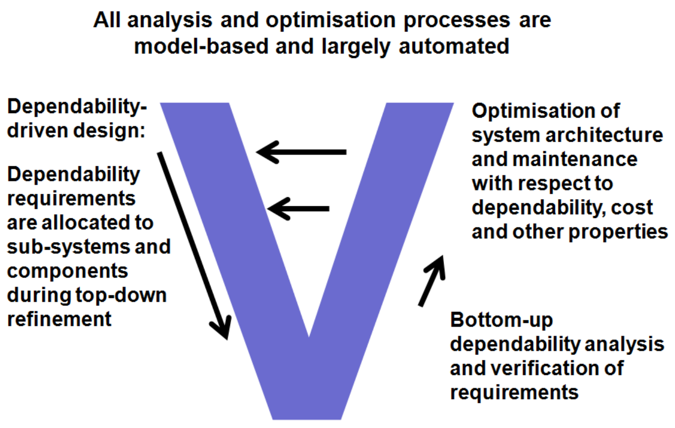

All analysis and optimisation processes are model-based and largely automated
Dependability-driven design:
Dependability requirements are allocated to sub-systems and components during top-down refinement
Optimisation of system architecture and maintenance with respect to dependability, cost and other properties
Bottom-up dependability analysis and verification of requirements

Fig. 1. Scope of HiP-HOPS in the V-lifecycle

At the early stages, HiP-HOPS supports a dependability driven mode of design in which system requirements are captured early and are allocated to sub-systems and components of the architecture. In a typical design the possibilities for allocation are numerous, so the process is partly automated via use of metaheuristics. The goal is to find an optimal allocation of system requirements to elements of the architecture within the space of all possible such allocations. Optimality here is defined mainly in terms of minimising the projected costs that would be associated with the different levels of integrity demanded from components to meet system requirements (Azevedo *et al.*, 2014), thereby assisting the designers in developing a solution that meets dependability requirements but minimising costs within the constraints of an established cost budget. This process can be repeated as the system design continues to evolve and is described in section 3.2.

As the architecture of the system is refined and more detailed models of the system are produced, more detailed qualitative and quantitative dependability analysis can also be performed. Such analyses may be used as evidence that the system requirements have been met. The process is automated via algorithms for synthesis and analysis of fault trees and FMEAs (Papadopoulos *et al.*, 1999 & 2011) and is described in section 3.3.

Moving to the right of the V-lifecycle, and assuming that dependability analysis shows that requirements cannot be met by the current system architecture, it is possible to initiate a process of architecture and maintenance optimisation in which the goal is to arrive at an improved architecture which meets dependability and other requirements with minimal additional costs. The process is once more driven by meta-heuristics and can be used to address problems such as the optimal selection of components and subsystems between available alternatives, decisions on the location and level of replication of components (Adachi *et al.*, 2011), and decisions on maintenance scheduling (Nggada, *et al.*, 2013). This is described in section 3.4.

Finally, in section 3.5, we also discuss how the HiP-HOPS approach can be used in conjunction with architecture description languages (ADLs), which provide an integrated framework and systematic methodology for the kind of V-lifecycle described here.

3.2 Dependability-driven Design Refinement with HiP-HOPS

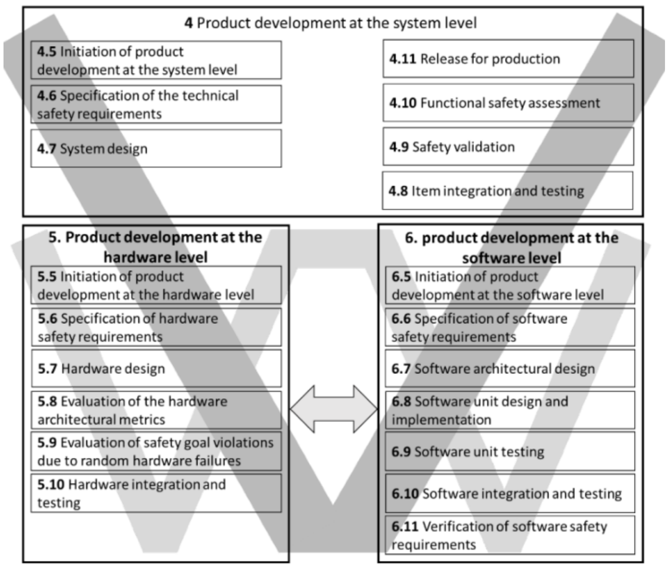
Many MBSA approaches are bottom up in that they rely on the prior existence of detailed system models that can be subjected to analysis or optimisation. As a counterpoint, we would like to pose a set of fundamental questions about system design.

a) Why should designers need to produce detailed designs before they can assess whether dependability requirements have been met, e.g. via MBSA?

b) Why should designers risk failing to meet requirements and then need to redesign?

c) Why not employ a top-down dependability-driven design process in which dependability requirements can be optimally allocated to sub-systems and components during refinement of the architecture?

The aspirations implied in the above questions concur with those expressed in modern safety standards. Using ISO 26262 (ISO, 2011) as an example, the standard defines a V-shaped safety lifecycle (shown below in *Fig. **2***). On the left of the 'V', safety requirements are established on the basis of a hazard analysis of the item being developed. These are then allocated in a top-down fashion to emerging system elements as the system architecture is continually refined from an abstract functional model to a more concrete hardware & software architecture. On the right of the 'V', the resulting architecture is then validated against the original requirements through a process of analysis and safety assessment.



4 Product development at the system level
4.5 Initiation of product development at the system level
4.6 Specification of the technical safety requirements
4.7 System design
4.11 Release for production
4.10 Functional safety assessment
4.9 Safety validation
4.8 Item integration and testing
5. Product development at the hardware level
5.5 Initiation of product development at the hardware level
5.6 Specification of hardware safety requirements
5.7 Hardware design
5.8 Evaluation of the hardware architectural metrics
5.9 Evaluation of safety goal violations due to random hardware failures
5.10 Hardware integration and testing
6. product development at the software level
6.5 Initiation of product development at the software level
6.6 Specification of software safety requirements
6.7 Software architectural design
6.8 Software unit design and implementation
6.9 Software unit testing
6.10 Software integration and testing
6.11 Verification of software safety requirements

Fig. 2. ISO 26262 V-model development process (ISO, 2011)

However, the guidance provided by these standards assumes requirements allocation is a manual process. We believe that an automated approach can better support the application of these standards and yield important improvements in the ongoing pursuit of improved design processes for dependable systems.

Standards such as IEC 61508 (IEC, 1998), ISO 26262, and ARP4754-A (SAE, 2010) introduce a system of classification for different levels of safety-integrity. IEC 61508 popularised the Safety Integrity Level (SIL), while ISO 26262 and ARP4754-A introduced domain-specific versions of this concept — the Automotive Safety Integrity Level (ASIL) for the automotive domain and the Development Assurance Level (DAL) for the aerospace domain. Integrity levels serve as a qualitative indication of the required level of safety or integrity of a function or component. Generally they are broken down into 5 levels, ranging from strict requirements (e.g. SIL4, ASIL D, DAL A) to no special requirements (e.g. SIL0, QM, DAL E). In some cases, quantitative targets are also associated with different levels, e.g. maximum failure rates for random hardware faults.

These integrity levels are employed as part of a top-down requirements allocation process as well as a bottom-up verification of those requirements. For example, as Fig. 2 shows, ISO 26262 describes a detailed safety process to be applied to automotive systems. The first step is a hazard analysis, which identifies the various malfunctions that may take place and the hazards that may arise as a result. The severity, likelihood, and controllability of these hazards are then considered. On the basis of this risk analysis, safety requirements — with associated ASILs — are applied to the various top-level functions of the system. The higher the risk of the function’s hazard, the higher the ASIL that is applied.

During the subsequent development of the system, traceability to these original ASILs must be maintained at all times. As the system design is refined into more detailed architectures, those original ASILs are allocated and decomposed to the subsystems and terminal components of the design. During the verification and validation, analyses (e.g. fault trees) must be produced to ensure that the refined system and — eventually — the final implemented system still meets the original requirements.

The process of allocating and decomposing ASILs across the architecture is far from straightforward. In general, the sub-components are responsible for providing the required integrity level of their parent. To avoid every element of the entire system having to meet the highest level of integrity, the different contribution of elements of the architecture to the numerous hazards predicted must be accurately established. Fault tolerant architectures (e.g. parallelism, redundancy, monitoring etc) can also be employed to spread the burden of meeting a high SIL across a number of components. While in some cases there is guidance on how this is to be done, achieving it in practice is often significantly more difficult. For example, ISO 26262 provides an "ASIL algebra", which indicates how a strict integrity level like ASIL D can be met by two independent subsystems which each individually meet lower ASILs, as shown in Table 1.

Table 1- Possible ASIL allocations for two subsystems

To meet ASIL D	Subsystem 1	Subsystem 2
Option 1	QM	D
Option 2	A	C
Option 3	B	B
Option 4	C	A
Option 5	D	QM

Note that there are many other possible combinations, but either these will not meet the overall ASIL D requirement or they will exceed it and thus incur unnecessary expense. The ASIL algebra can be thought of as a set of integer values for each ASIL (see Table).

Table 2- The ASIL algebra

ASIL	Value
QM	0
A	1
B	2
C	3
D	4

Assigning ASIL C and ASIL D to both subsystems would meet the overall requirement, i.e., $3 + 4 \geq 4$, but is inefficient. Assigning ASIL A and ASIL B would not meet the overall requirement, i.e., $1 + 2 < 4$. The most efficient allocations precisely meet the overall requirement, and these are shown in Table 1.

Thus there are problems in ensuring that constraints about components' failure independence are met, working through the many possible allocations, and ensuring that the decomposed low-level requirements still add up to the original high-level requirements. When dealing with the types of detailed electronic architectures that are common in modern safety-critical systems, following safety standards like ISO 26262 and ARP4754-A manually with respect to SIL allocation requires additional time and expense and is often infeasible. Even when considering only two subsystems, there are 25 (i.e., 5^2) possible ASIL allocations, of which only 5 are 'optimal' in the sense that they precisely meet the overall requirement. The scope of the problem only grows when more subsystems and more system-wide requirements are considered.

Standards are naturally focused on safety and SIL allocation is presented as a problem where the single goal is to find an allocation of integrity requirements to components of the system architecture that meets system level integrity targets. HiP-HOPS, however, explores an additional dimension of the problem, which is very relevant for developers of systems: cost. Consider the earlier example where there are 5 different options to decompose an ASIL D across two redundant components. Which one should be selected? While all of the options are valid with respect to ISO 26262, some will likely be more costly — typically those assigned high ASILs such as C and D. Clearly, the allocation that minimises SIL-dependent costs is preferred, and in this case that might mean allocating ASIL B to both subsystems. Table shows how an abstract measure of cost can be observed using a simple cost heuristic in which QM has a cost of 0, ASIL A has a cost of 1, ASIL B a cost of 10, C a cost of 100, and D a cost of 100.

Table 3- Allocating ASILs with a cost heuristic

Subsystem 1	Subsystem 2	Overall cost
QM (cost = 0)	D (cost = 1000)	0 + 1000 = 1000
A (cost = 1)	C (cost = 100)	1 + 100 = 101
B (cost = 10)	B (cost = 10)	10 + 10 = 20
C (cost = 100)	A (cost = 1)	100 + 1 = 101
D (cost = 1000)	QM (cost = 0)	1000 + 0 = 1000

When considering an isolated decomposition decision, as in the example above, this might seem like a trivial problem of comparing the costs between a few options. Components, however, often participate in assuring multiple functions, and when trying to find cost-optimal solutions, numerous chains of conflicting constraints involving many components need to be taken into account. Furthermore, as the number of components increases, the number of allocation combinations to examine grows, often exponentially. This is a complex combinatorial problem where the satisfaction of integrity requirements from safety standards is a constraint that must be met but where the real objective is the minimisation of development costs (Azevedo *et al.*, 2014).

In HiP-HOPS, a largely automated process for SIL allocation and decomposition is available, which is built on the dependability analysis framework presented in section 3.3. HiP-HOPS can establish the potential contribution of combinations of component faults to system level failures. From this information, it is then possible to automatically perform a allocation and decomposition of requirements. HiP-HOPS takes into account the component dependencies and can therefore automatically apply the type of SIL 'algebra' described by ISO 26262 and ARP4754-A. This allows SIL allocations to individual components, component ports, or even component failure modes. Allocating SILs to failure modes allows a more efficient refinement of requirements because when a component can fail in multiple ways, only the sub-components causing its most severe failures are assigned the higher SIL; sub-components causing less serious failures can receive lower integrity levels.

Work in this area has shown that the number of different potential allocation schemes that can meet a complex set of system requirements often produces a vast search space, and exploring this exhaustively with deterministic optimisation algorithms for a cost-optimal allocation can be problematic (Parker *et al.*, 2013). Recent work has therefore focused on the use of metaheuristics to efficiently explore this large space. The resulting allocations of SILs meet the system requirements with significantly reduced costs for procurement, development and verification of components. We have applied and evaluated a range of metaheuristic optimisation algorithms in this context:

- (Parker *et al.*, 2013) applied genetic algorithms to allocate SILs in an automotive hybrid braking system. Genetic algorithms rely on processes of natural evolution to find promising solutions and incorporate a random "mutation factor" to avoid being caught in local optima. The search space in this instance was 5.96 x 10^{16}, small enough to exhaustively determine that there were 125 optimal allocations. 4 out of 10 runs found the optimal solutions, while the other 6 runs found non-optimal but very low cost solutions (within a percent or two of optimal). Each run took a few seconds.
- In (Azevedo *et al.*, 2013), Tabu Search was applied to the same hybrid braking system according to a variety of different cost heuristics. Tabu search is a metaheuristic optimisation method which utilises memory structures known as the Tabu Tenure, applied over a local search heuristic. The Tenure allows memorisation of recently found candidates which can then be avoided in some of the search's upcoming iterations, allowing more opportunities to explore new regions of the search space and helping to avoid being trapped in local optima. 9 out of 10 runs provided optimal solutions for all cost heuristics and each run took only a few milliseconds.
- In (Sorokos *et al.*, 2015), a Tabu search-based approach was applied to an aircraft wheel brake system, using DALs and the different decomposition logic set out in ARP4754-A. In this case every run of the algorithm produced an optimal solution, although the search space was considerably smaller.

More recently, we have also begun to investigate Particle Swarm-based approaches. These use a strategy to explore the solution space based on the social behaviour of flocks of birds, or schools of fish, in their search for food. The position visited by a particle represents a candidate solution; when choosing the next position to visit, a particle is influenced by the best position it has so far visited and by the best position encountered by its neighbours.

As shown in the above papers, this work has produced promising results that indicate good scalability. Prototype implementations of the different metaheuristics have been incorporated into experimental versions of the HiP-HOPS tool and even the slowest (genetic algorithms) can find optimal allocations for medium-sized problems in a few seconds. Exhaustive search methods and techniques like integer linear programming have been shown to take longer but with the trade-off that they find a set of solutions that are guaranteed optimal (Murashkin *et al*, 2015). In all cases, even when the costs are not optimal, the allocations found always meet the overall safety requirements.

In most cases, HiP-HOPS can now calculate optimal allocations of dependability requirements to subsystems and components of a system, taking into account their dependencies and assumptions about their intended behaviour in conditions of failure. Stakeholders in a value chain are able to apply this capability iteratively in order to specify procurement contracts for the development or purchase of sub-components that define increasingly refined dependability requirements to suppliers in lower tiers of a value chain. The process guarantees that a system will meet its dependability requirements at the end of the design process providing the various contracts, such as requirements on basic components; assumptions of independence etc., have been met

The concept is recursive and can be applied in exactly the same fashion between any two tiers of a value chain. It is independent of industry and can incorporate different rules and algebras, which makes it compatible with a variety of contemporary standards from different domains.

Note that at this stage, the process of SIL decomposition and allocation does not involve any changes to the architecture itself: we are only allocating requirements to existing sub-elements of the model, on the basis of overall safety requirements derived from system-level hazards. Later, if these requirements are shown not to be met by the refined system design, we can modify the architecture accordingly, as described in section 3.4.

3.3 Modelling and Dependability Analysis

At various stages through the V-lifecycle, it is useful to be able to perform dependability analysis on the system architecture — whether an initial abstract feature model, a more refined functional architecture, or a concrete hardware/software architecture — to obtain information about the possible causes of failures in the system and how likely those failures are. This information can be used to verify that decomposed safety requirements are being met and, if performed iteratively, can also inform the direction of the design by highlighting potential weakpoints in the system.

All analysis and optimisation processes in HiP-HOPS are performed on an architectural system model which identifies material, energy, and data transactions among components (Fig. 3).

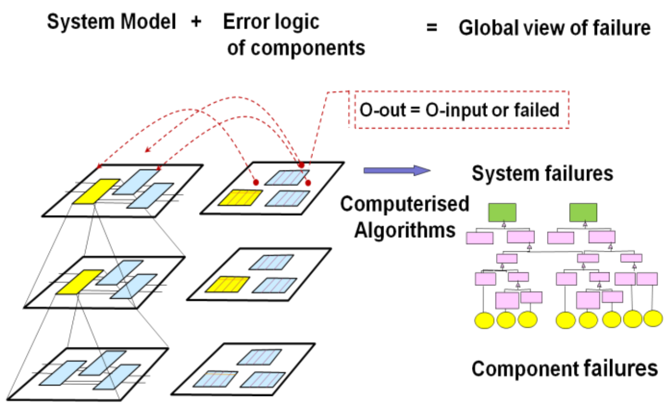

System Model + Error logic of components = Global view of failure
O-out = O-input or failed
System failures
Computerised Algorithms
Component failures

Fig. 3. Modelling and Dependability Analysis in HiP-HOPS.

The model can be hierarchical if necessary to manage complexity. In the case of a hierarchical model, subsystems enclose architectures of more basic subsystems and components.

The dependability analysis process in HiP-HOPS then proceeds in three phases. The first phase is the *annotation phase*: each component in the model is annotated with its local error logic, describing the errors that can occur in the component and how it responds to deviations of its inputs. HiP-HOPS defines a language for the description of this error logic. In the basic version of this language, the error logic of a component can be specified as a list of internal failure modes of the component and a list of errors or deviations as they can be observed at component outputs. Each component failure mode is optionally accompanied by quantitative data, for example a failure and a repair rate. Output errors carry Boolean expressions which describe their causes as a logical combination of component faults and similar errors observed at component inputs. For example, to describe an omission of output from a component caused by either an omission of corresponding input or an internal failure mode, we can say:

```
omission-component.outputPort =
     internalFailure OR
     omission-component.inputPort
```

Here `internalFailure` is a failure mode of that component and may have probabilistic failure data attached, e.g. a failure rate in terms of failures per hour. `omission-component.inputPort` represents an error or deviation at the component's input port of type "omission", i.e., a lack of input. The logical OR operator indicates that the output deviation — omission of output — is caused by either the internal failure mode or the lack of input to the component.

Input and output errors referenced in the error logic are described qualitatively and typically represent different classes of failures, such as the omission or commission of parameters or qualitative deviations from correct value (i.e. hi/low) and expected timing behaviour (i.e. early/late). These are not fixed and analysts may use whatever nomenclature they wish as long as the usage is consistent across the model.

Collectively, a set of failure expressions that logically explain all possible errors at all output ports of a component provides a model of the error logic of the component under examination. This model can be stored in a library. For simple components, e.g. sensors and actuators, such models could be re-used across different applications to simplify the manual part of the analysis and the overall application of the proposed technique.

The second phase of the HiP-HOPS dependability analysis process is the *synthesis phase*. Using the error logic associated with components, computerised algorithms automatically determine how errors propagate through connections in the model to cause functional failures at system outputs. These are the failures that analysts are typically interested in identifying and analysing. For example, in a car, such functional failures may include the loss of steering or braking. Since HiP-HOPS shows how individual failure modes in components can combine and lead to functional failures at system outputs, a system failure such as loss of braking may be seen to be the result of an actuator failure.

This global view is captured in a set of interconnected fault trees. These fault trees show how the leaf nodes of the trees — representing the component failure modes and their local effects — can logically combine and propagate though the system to cause the top events of the fault trees, which represent the functional failures of the system (Veseley *et al.*, 2002). The interconnections between the trees represent dependencies in model, e.g. the failure of a common power supply or a global condition that may affect more than one system function. Common cause failures, such as flooding of physically co-located components, can also be represented in HiP-HOPS.

Once this is done, the third phase of HiP-HOPS is to perform analyses of this global system error model: the *analysis phase*. First, an automated fault tree analysis is performed for each of the functional failures in the system. HiP-HOPS can perform both qualitative and quantitative analysis of fault trees. Qualitative analysis is used to establish the minimal cut sets of the fault trees — the smallest combinations of failure events necessary to cause system failure — which more readily indicate how system failures may occur. Quantitative analysis is also possible when probabilistic parameters have been provided at component level and is used to predict the reliability and availability of the system.

In the final stage of the analysis, the complex body of logic encoded in the set of interconnected fault trees is simplified by an automated algorithm which translates it into a simple table of direct relationships between component and system failures. In a similar way to a classical FMEA, this table determines, for each component in the system and for each failure mode of that component, the effect of that failure mode on the system. The table shows which system failures (if any) each failure mode causes, both by itself and in conjunction with other events.

Note that in a classical manual FMEA only the effects of single failures are typically assessed. Thus, one advantage of generating an FMEA from fault trees is that fault trees record the effects of combinations of component failures and this useful information can also be transferred into the FMEA. The FMEA shows all the functional effects to which a particular component failure mode contributes, both individually and as part of a combination. This is particularly useful as a failure mode that contributes to multiple system failures is potentially more significant than those that only cause a single top event. Consequently, this type of FMEA can also help analysts to determine the level of fault tolerance in the system, i.e., to determine whether the system can tolerate any single failure or any combination of two, three or more component failures.

It is clear that both quantitative and qualitative analyses in HiP-HOPS can play a dual role: either to help verify requirements or stimulate useful design iterations by highlighting weak areas of the design.

We should note that experimental versions of the tool enable use of an extended language where it is also possible to express a wider range of failure semantics. For example, wildcards can also be used to describe more abstract patterns of relationships between output and input deviations. This allows statements such as "there will be an omission of all outputs in response to any input error" (Wolforth *et al.*, 2010), which assists in the reuse of error logic descriptions across components with different interfaces but similar failure behaviour.

More significantly, recent work has extended the range of systems that can be effectively analysed by HiP-HOPS. Because it is based on classical Boolean fault tree analysis algorithms, traditionally HiP-HOPS was limited to analysing only those systems that could be represented with Boolean failure behaviour. However, many safety-critical systems exhibit more complex behaviour: they may be dynamic, rely on sparse or uncertain probabilistic data, or express non-coherency in their failure logic.

Experimental versions of HiP-HOPS provide support for all of these types of scenarios, as will be explained next.

3.3.1 Dependability Analysis of Non-Coherent Systems

Some safety critical systems exhibit non-coherent failure behaviour, which means that certain system failures can only occur if another event has *not* occurred. This other event may be an ordinary system event or it could be another failure event. Such scenarios may occur in multitask or multi-phase systems where the causes of system failures can only be identified once the system successes have been taken into account (Andrews, 2000). For example, a failure in task A may only occur if another task B has succeeded: failure of task B therefore prohibits failure of task A. In fault trees, this condition is typically represented using a NOT gate.

HiP-HOPS has been extended with the capability to model and analyse this type of scenario (Sharvia & Papadopoulos, 2008). HiP-HOPS failure expressions can include NOT gate conditions so that the effects of failures not occurring can also be taken into account during system analysis. This involves the generation of the prime implicants, i.e., the effects of different failure states of multiple components in combination, at the expense of a slight performance overhead. The resultant fault trees and FMEA tables then show that some system failures may only occur if certain system conditions or events do *not* occur.

3.3.2 Dependability Analysis of Dynamic Systems

In a dynamic system, the system behaviour changes over time. This could be because there are multiple phases of operation, e.g. as in an aircraft with distinct take-off, flight, and landing phases, or it could be because the system behaviour changes in response to different events (whether normal system events or failure events). Safety-critical systems are increasingly dynamic in nature as they frequently include the capacity for partial self-repair in response to failure, e.g. through the use of backup components, fallback to degraded modes of operation, or automatic detection and correction of certain types of errors.

Classical safety analysis techniques such as FTA and FMEA struggle to model these types of scenario. The key shortcoming is the inability to distinguish between the effects of different *sequences* of events, not just combinations of events.

Consider the triple redundant system of Fig. 4:

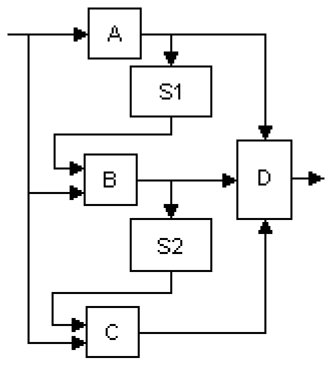


Fig. 4. Dynamic system with two backup components.

System functionality is initially provided through component 'A', which is the primary component. The monitor 'S1' observes component A for any deviation of its outputs. If detected, the sensor activates component 'B', the first backup component, and moves to a degraded mode of operation. Similarly, once component B has been activated, monitor 'S2' begins monitoring B for output deviations and, if any are detected, activates the final backup component 'C'. Component 'D' represents the system output.

A standard FTA of this system might indicate the following causes of system failure:

1. Omission of input to the system
2. Failure of A *and* Failure of S1 to detect failure of A

3. Failure of A *and* Failure of B *and* Failure of S2 to detect failure of B
4. Failure of A *and* Failure of B *and* Failure of C

At first glance, these causes would seem to make sense. However, they do not take into account the sequence of events. For example, consider the second cut set: if component A failed first, allowing S1 to activate component B, a subsequent failure of S1 would have no further effect on the system — it is already running in its degraded mode using backup component B. Therefore this cut set is pessimistic, as only one sequence (failure of S1 *before* failure of A) will lead to system failure. Similarly, a failure of component B before component A means that B will never be activated by S1, and thus monitor S2 will never be able to activate the final backup component C. In this case, the last cut set is optimistic, because some sequences of events will result in system failure even without the failure of C.

To remedy this general problem, a range of temporal and dynamic extensions to fault trees have been proposed, such as the Dynamic Fault Tree approach (Dugan *et al.*, 1992; Veseley *et al.*, 2002). For HiP-HOPS, we have developed an extension to the fault tree analysis known as Pandora (Walker *et al.*, 2007). The Pandora technique is included as part of HiP-HOPS and adds new temporal logic gates to enable fault trees to model sequences of events and thus better capture the failure behaviour of dynamic systems.

In particular, Pandora adds three new gates:

Priority AND gate (PAND):	`X < Y`
Simultaneous AND gate (SAND):	`X & Y`
Priority OR gate (POR):	`X	Y`

The PAND gate represents a sequence: event X must occur before event Y, but both must occur. The SAND gate represents simultaneous occurrence. The POR gate represents a condition: event X must occur before event Y if event Y occurs at all.

By defining a set of new temporal laws that apply to these gates, analogous to the laws of Boolean logic, Pandora makes it possible to perform a qualitative analysis of temporal fault trees and obtain the minimal cut sequences — the smallest sequences of events necessary to cause the system failure.

Using Pandora, the minimal cut sequences of the example triple redundant system would be as follows:

1. Omission of input to the system
2. Failure of S1 *before* Failure of A
3. Dormant failure of B *before* Failure of A
4. Failure of S2 *before* Failure of B *or* A
5. Failure of A *before* Failure of B *and* Failure of C

These cut sequences better capture the dynamic behaviour of the system. The failure of a monitor after the failure of the monitored component is no longer modelled as a system failure and the scenario where B can fail dormant before A is properly represented.

In the absence of quantitative data, Pandora can provide useful insight into system failure. Where quantitative component failure data is available, quantitative analysis of Pandora fault trees is possible. The analytical approach (Edifor *et al.*, 2012, 2013) uses mathematical expressions to probabilistically evaluate the Pandora temporal fault tree gates based on exponentially distributed failure data. Using this approach, analysts can determine the overall reliability of a system. They can also identify the critical parts of the system by determining the relative contributions of the various system components to the causes of system failure. Once the critical parts are identified, reliability may be improved by e.g. including redundant components or using components with lower failure rates.

Petri Nets (PNs) have also been used to develop an approach for probabilistic evaluation of Pandora fault trees based on exponentially distributed data (Kabir *et al.*, 2015). In this approach, the fault trees are quantified by translating them into Generalised Stochastic Petri Nets (Marsan *et al.*, 1996). Similar to the analytical solution, this approach can evaluate system reliability and criticality of components. In addition, it can also verify the correctness of the qualitative analysis. To allow the analysts to perform quantitative analysis of Pandora fault trees with any kind of distributions of data, Kabir *et al.* (2014a) have developed a methodology based on Bayesian Networks. Although others have developed approaches to convert Boolean fault trees to Bayesian Networks (Bobbio *et al.*, 2001), this approach instead transforms Pandora temporal fault trees to evaluate system reliability and criticality of system components. In addition to the predictive analysis, this approach allows the analysts to perform post-hoc diagnostic analysis, a process which involves calculating and updating the posterior probability of basic events given observed evidence of the system failure.

3.3.3 Dependability Analysis in Conditions of Uncertainty

As already mentioned, HiP-HOPS can perform quantitative analysis to predict the reliability and availability of systems if the probabilistic parameters of system components (e.g. failure rate or failure probability) can be provided. However, this means that the quantitative analysis is entirely dependent on the availability of this quantitative failure data. For many complex systems, it is often difficult to obtain precise failure data of components from past occurrences due to lack of knowledge about the systems, scarcity of statistical data, and changes in operating environment of the systems (Tanaka *et al.*,1983; Singer, 1990). This situation is particularly relevant in the early design phases when system analysts consider new or undetermined components for which there is no quantitative data. In such situations, expert human judgement in linguistic terms, e.g. 'very low, low, high' may be used to determine uncertain failure data of components.

Fuzzy Logic is a branch of mathematics which has the capability to deal with linguistic variables and it provides efficient way to draw conclusions from imprecise data. A variety of approaches (e.g. Suresh et al. (1996); Yang (2012)) have been proposed based on fuzzy set theory to allow classical and dynamic fault tree analysis with uncertain data. Recently, a fuzzy set theory based methodology has been proposed by Kabir *et al.* (2014b) to quantify Pandora fault trees with uncertain data. In this method, fuzzy operators for the new fault tree gates have been developed and fuzzy data have been used in the quantitative analysis instead of fixed data. As a result, the system unreliability is obtained as fuzzy numbers. This methodology can also determine the criticality of system components based on their relative contributions to the occurrence of the system failure. By more explicitly highlighting the areas of uncertainty in the failure data, this method can lead to a more effective quantification of uncertainty in dynamic systems. It is important to highlight that the results of quantitative analysis can only be as reliable as the input data, and the inclusion of fuzzy data cannot improve the accuracy of the results. However, techniques such as importance measures allow analysts to see the relative contribution of different system elements to the overall failure probability. This helps to overcome the limitations of uncertain quantitative analysis results by focusing on the relative values rather than exact values, identifying the areas of the system design most sensitive to improvement.

3.4 Architecture and Maintenance Optimisation

Let us assume now that a team of analysts is designing a system, that we have decomposed the dependability requirements across the architecture, and that an analysis from a MBSA tool suggests that the system does not meet all of those dependability requirements. At this stage we need to improve the design somehow so that it does meet the requirements, as shown by a second round of analysis. There is typically a range of options available to improve a design, including:

a) replacing a component with a more reliable and expensive component

b) replacing part of the architecture with a more dependable alternative

c) replicating components in fault tolerant schemes so that failures are tolerated

d) increasing the frequency of maintenance, an action that prolongs the useful life of components and thereby increases the reliability of the system.

The difficulty is that in a typical system design, there is a very large number of possibilities for substitution, replication and maintenance scheduling. For instance, in a system of n components, if there are two suppliers for each component then there are 2^n configurations which equates to $1.26e^{30}$ configurations when n=100. Each configuration will have its own dependability and cost performance. It is clear that in such situations analysts are confronted with a multi-objective optimisation problem, where the objectives may include dependability, cost, weight and other properties.

It would be prohibitively expensive to investigate more than a handful of these possible configurations manually. Therefore, to optimise such designs, we have developed an extension of HiP-HOPS that employs genetic algorithms to perform multi-objective optimisation of architectures with respect to dependability and other attributes (Papadopoulos and Grante, 2006; Papadopoulos *et al.*, 2011). This is a separate optimisation process to the allocation of dependability requirements; architectural optimisation takes place as a way of finding a design that meets the devolved dependability requirements, which may themselves have been set as a result of an allocation optimisation process.

The architectural optimisation concept is illustrated in Fig. . As with dependability analysis in HiP-HOPS, the process starts from a model of the system. However, this time the model is not fixed — it has *variability*, i.e., components can have multiple alternative implementations. These points of variability may involve different parameters of components or may involve architectural changes, e.g. replacing a single component with a more fault-tolerant design using primary and backup components. For example, a sensor can be chosen from two different suppliers, with each choice having its own cost, weight, performance, and failure characteristics. Subsystems can also carry alternatives, e.g. a subsystem can have two different implementations that provide the functions using different sets of components and different architectures. There can be options for replication of components with known patterns of fault tolerance, e.g. a primary-standby configuration, or multiple parallel channels with majority voting. Finally, there can be options for the scheduling of component maintenance.

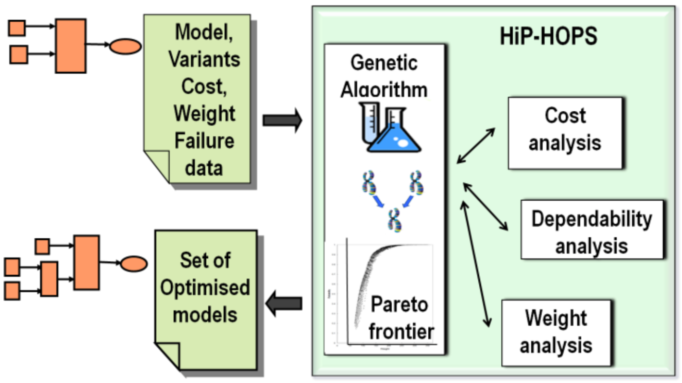


Fig. 5. Architecture optimisation in HiP-HOPS.

Once the system model has been annotated to include these variable possibilities and any further required information, including associated cost and failure data etc, the model is given to HiP-HOPS, which then applies an evolutionary optimisation process. In the context of this process, HiP-HOPS creates a population of candidate designs by resolving the variability of the model, i.e., fixing variation points in the model by selecting particular design options. Each candidate design is then evaluated with respect to the objectives of the optimisation. The evaluation is performed using the analysis algorithms of HiP-HOPS. The reliability and availability of a candidate design are automatically calculated from the generated fault trees. A quantitative measure of safety is established from the FMEA, using, for instance, the number of single points of failure that contribute to severe system failures. HiP-HOPS also includes simple summative cost and weight functions. External plugins can also be designed to enable more precise evaluation of cost, weight or other objective functions. For example, experiments with timing and schedulability have been reported in (Walker *et al.*, 2013).

Once candidate designs have been evaluated, they are ranked according to their performance and a Pareto frontier is formed showing the best designs in the current population. Roulette wheel selection, a random process biased towards the better performing designs, is used to select candidates to form the parents of the next generation. Through application of classic genetic operators such as mutation and crossover, a new population is then formed and the process of evaluation and ranking is iterated. The result of this process over a number of successive generations is a gradual improvement of the average performance of the population that is evident in the progressive improvement of the Pareto frontier. The process is terminated on meeting certain constraints or after a specified number of generations. The result is a set of models that give optimal or near optimal trade-offs among the objectives of the optimisation.

Via this process, designers can take informed decisions about the selection of components, subsystems, the location and type of replication, and about maintenance scheduling, all the while making sure that dependability requirements can be met whilst minimising costs.

As an example of this architectural optimisation process, HiP-HOPS was applied to a high-level abstract design of a vehicle pre-collision system and an evolutionary optimisation technique was used to achieve balanced solutions with respect to dependability and cost (Adachi *et al*., 2011).

The pre-collision system is an automotive safety technology that avoids or reduces the damage caused by a collision. The system supports drivers by issuing warnings when a potential collision threat is identified and activates emergency braking if the driver fails to apply the brakes. To improve system fault tolerance, a number of fault tolerance mechanisms were considered. These mechanisms may be applied to various locations in the system architecture to achieve greater dependability, albeit at an increased cost. The mechanisms include *self-protection, self-checking, checkpoint-restart and process-pair*. *Self-protection* and *self-checking* are functions which can be used for error detection. In *self-protection*, the component protects itself from external disturbances by detecting errors propagated from other components. In *self-checking,* a component detects internal errors and prevent the propagation of those errors to other components. *Checkpoint-restart* not only detects failures, but also recovers from errors by restarting the component. Finally, *process-pair* is a fault tolerance technique which uses redundancy realised by two identical software components. These are typical mechanisms for detection and correction of errors which give a sophisticated range of options to consider in early design. To model situations where these fault tolerant components miss some failures which need to be detected, an additional event *miss* was included in the analysis. Fault tolerant mechanisms may also experience failure, so the event *failure* is used to represent internal malfunction for the fault tolerant components. Information on failure expression and failure rate were included for each of the components in the system with reasonable assumptions about plausible hardware and software failures. The HiP-HOPS optimisation algorithm was finally employed to select the optimal location and types of fault-tolerance mechanisms in an improved version of the system. From a total design space of about $12^7 \approx 3.6\times10^7$, in just 5 minutes it was able to find 8 Pareto optimal solutions that provided a good trade-off between risk and cost while meeting the required constraints (see Adachi *et al* (2011) for further information).

The case study showed that insight into the optimal use of fault tolerance can be arrived at much more rapidly with the aid of automated tool support. The vast number of different options, let alone the time required to evaluate and compare these options, would make an equivalent manual process infeasible. Thus metaheuristic approaches allow a designer to obtain significant improvements in reliability and cost performance.

3.5 Model transformations from Architecture Description Languages

HiP-HOPS has also been used to support the development and analysis of systems modelled using Architecture Description Languages (ADLs), particularly the Architecture Analysis & Design Language (AADL) (www.aadl.info) and the automotive EAST-ADL (www.east-adl.info).

EAST-ADL provides an integrated and systematic support for the modelling of automotive systems. The growing adoption of model-based engineering techniques like EAST-ADL is driven by the need to better-manage advances in functionality and corresponding increases in the complexity of modern safety-critical embedded systems. The specification of EAST-ADL includes an error model which describes potential failures of design elements.

To enable advanced analysis capabilities like FTA, FMEA, optimisation and safety requirement allocations, the EAST-ADL error model is extended with HiP-HOPS semantics. This integration requires translation of models in the automotive domain to models in the safety analysis domain, i.e., a transformation of an EAST-ADL error model to a corresponding HiP-HOPS model. The concrete source and destination models are both represented in XML-based formats, which are EAXML and HiP-HOPS XML respectively. The translator tool is described in (Sharvia *et al.*, 2014). It involves conceptual semantic mapping between the domains and the representation of concrete models.

The benefits of ADL's such as EAST-ADL and AADL depend crucially on the availability of tools. Model transformation has been used make the optimisation capabilities of HiP-HOPS available to AADL models (Mian *et al.*, 2014). At the highest level of abstraction, the transformation consists of two parts. One part is concerned with the component specific error behaviour and the other part is concerned with the inter-component error propagation.

AADL uses an Error Model Annex for modelling component failure behaviour. Error models in AADL are state machines which specify how the state of a component changes in response to events or the states of other components. The model transformation incorporates a state machine to fault tree conversion algorithm described in Mahmud *et al*. (2012). This preserves the temporal properties captured in the state-machine.

The Atlas Transformation Language (ATL) (Jouault *et al*., 2008) is used to implement the transformation which has been developed as a plug-in for the AADL model development tool OSATE (https://wiki.sei.cmu.edu/aadl/index.php/Osate_2, accessed 2015).

4. TECHNICAL DISCUSSION

Key to all of the approaches presented in section 3 is the underlying system model in HiP-HOPS. At its core this is a architectural model that shows system elements and possible data, material, or energy flows between them. In the HiP-HOPS tool, this model can be exported from widely-used system modelling packages including Matlab Simulink (Mathworks, 2016), SimulationX (ITI, 2016), and various Eclipse-based UML modelling platforms such as Papyrus (Eclipse Foundation, 2016). As described in section 3.3, this model is further annotated with logical descriptions of the local failure behaviour of system elements. HiP-HOPS can then use this information to build a failure propagation network, describing how failures propagate through the system and revealing the dependencies between the different system elements. Because HiP-HOPS models are generated automatically from existing engineering models, it is easy to make modifications to the actual system model and then very quickly observe the effect this has on the analysis results.

Similarly, to support the different optimisation processes, HiP-HOPS requires information about the different possibilities. For decomposition of dependability requirements, it requires data on the system-level functional safety requirements, the cost heuristic to be used, and also what SIL algebra is to be used. For architectural optimisation, more detailed information is required in the form of different alternative implementations for each component to be used as a variability point, whether in the form of different parameters (e.g. cost, weight, reliability, maintenance schedules) or different sub-architectures (e.g. series, parallel, fault tolerance schemes).

Clearly, it is important for the model and its associated information to be correct. The failure propagation model is used to ensure independence between decomposed sub-elements of the system, without which the analysis results will be in error and the allocation of dependability requirements will be invalid. Consequently, we have made efforts towards improving the expressiveness of the HiP-HOPS model and its annotations, including modelling dynamic behaviour as explained in section 3.3.2 and for uncertain data as in 3.3.3.

However, due to the fact that analysis takes no more than a few seconds even for large systems with hundreds of components and many thousands of cut sets, it is relatively easy to identify errors, correct them in the model, and regenerate the analysis results compared to the effort that would be required to repeat a full manual safety analysis. The HiP-HOPS tool also performs a range of checks and reports various warnings and errors when it detects potential errors in the modelling or the failure annotations.

There are other limitations to our current work, typically consequences of being based on an easy-to-use Boolean logic rather than more complex state-based approaches. Logical loops in the failure propagation can be problematic (though are often symptomatic of modelling errors) and for this reason the tool works better at higher levels rather than on low-level electronic circuits. Repairable components are supported by the core safety analysis but not by all of the experimental extensions to the tool. Nevertheless, we are continually undertaking further work to try to address many of these limitations.

Ultimately any analysis is only as good as the data it is based on, and we rely upon the designers to provide accurate failure data for their system models.

5. RELEVANT WORK

There is very little work reported in linking MBSA to metaheuristics. In (Konak *et al.*, 2007) systems are represented as Reliability Block Diagrams (RBDs) which are subsequently optimised using meta-heuristics. HiP-HOPS supersedes this work by enabling optimisation of models which may have a networked architecture, i.e. they are not necessarily in parallel or series configurations as RBDs, and overcome the traditional assumption made in RBDs that a component or system either works or fails in a single failure mode. HiP-HOPS has been the first approach to direct optimisation of dependability on an architectural model. Other tools for architecture optimisation, with the possibility of adding arbitrary quality properties as objectives, include ArcheOpteryx (Aleti *et al.*, 2009) and PerOpteryx (Koziolek, 2011). The scope of these tools includes architecture optimisation but does not include the requirements allocation problem.

These tools require a reliability evaluation model such as a fault tree, RBD or Markov Chain for evaluating reliability. HIP-HOPS re-synthesises this model during the evolution of the system architecture by operating directly on an architectural model augmented with failure data. HiP-HOPS has also incorporated the first effort directed towards automatic allocation of dependability requirements (Papadopoulos *et al.*, 2010) and remains the only application of metaheuristics in this area. Mader *et al.* (2012) proposed an approach for ASIL allocation where a linear programming optimisation problem is formulated to discover a solution that minimises the sum of ASILs as-signed across the system architecture. Zhang *et al.* (2010) proposed a workflow for embedded system development, which includes fault trees, FMEA and ASIL allocation based on a qualitative risk graph method. Dhouibi *et al.* (2014) introduced a method for ASIL allocation which is based on interpreting the allocation problem as a system of linear equations. Bieber *et al.* (2011) presented a theory to formalise the ARP4754-A DAL allocation rules and the DALculator tool to support automatic DAL allocation via integer programming optimisation. The starting point for these approaches are minimal cut sets of fault trees. Instead, HiP-HOPS starts from architectural models, offering the advantage of being able to assess explicit or implicit dependencies in the model and its environment that may cause common mode failures.

6. CONCLUSIONS

The technologies of model-based design, dependability analysis and the application of heuristics to the design of dependable systems, including software intensive systems, have advanced in recent years. However, we have not yet seen the emergence of a design paradigm that employs these techniques synergistically and systematically from the early stages of design to enable cost-effective, dependability-driven optimal design refinement.

In this paper, we have outlined four challenges that remain unaddressed and sketched a model-centric paradigm for the design of dependable systems that brings these technologies together to realise their potential benefits. These benefits include:

- controlling dependability from the early stages via optimal allocation of requirements;
- effective top-down distribution and then bottom-up composition of dependable designs in collaborative environments, distributed across complex value chains;
- automation in the assessment of design proposals and prediction of dependability;
- decision support on optimisation of architectures for component selection, fault tolerance and maintenance scheduling;
- reuse of repositories of models and analyses both during design refinement and across projects.

Tackling the wide range of requirements to obtain these benefits requires a model-based design paradigm that draws upon state-of-the-art developments and knowledge from multiple fields, building on classical and temporal logic, biology-inspired metaheuristic techniques and modern model-based engineering principles. In this paper, we have shown that such a paradigm is feasible by discussing its embryonic incarnation within the HiP-HOPS method and tool. HiP-HOPS is presently the only MBSA method that applies metaheuristics across the lifecycle including the very early stages, addressing both requirements and architecture. The transferability of this work in model-based design has been demonstrated in the context of architecture description languages such as EAST-ADL (Walker *et al.*, 2013) and AADL (Mian *et al.*, 2014).

We do not claim that we have addressed the enormous challenges discussed in section 2. Our modest aim was to show that this synthesis of bio-inspired techniques with logic has the potential to improve the field of MBSA by enabling useful functionalities that were previously unexplored. This is where we see the value of the paper and we hope that this modest claim has been substantiated. Our experiments, which are described in many of the references, show practical improvement in design using these functionalities. One can see Pareto fronts of generated solutions moving towards better and better tradeoffs. We have still not attempted a systematic quantification of these improvements. One way to achieve this is by tasking engineers with developing solutions to problems also solved with the aid of metaheuristics. One could then plot these solutions on Pareto fronts and could measure the distance in performance between manually derived and automatic solutions.

There are of course many other challenges that remain to be addressed as this work develops further within the field of model-based design and MBSA. These include the representativeness and completeness of models, the relation of models to code, the modelling and analysis of commercial off-the-shelf or legacy systems, the efficacy of automatic model-transformations in the context of optimisation and the scalability of models with respect to computational cost of analyses.

REFERENCES

Adachi M., Papadopoulos Y., Sharvia S., Parker D., Tohdo T. 2011. An approach to optimization of fault tolerant architectures using HiP-HOPS, *Software Practice and Experience*, 41:1303-1327, Wiley.

Aleti A., Bjoernander S., Grunske L., Meedeniya I. 2009. ArcheOpterix: An extendable tool for architecture optimization of AADL models, *ICSE 2009*: 61-71.

Andrews J.D. 2000. To Not or Not to Not. In *Proceedings of the 18th International System Safety Conference,* September 11-16, Fort Worth, USA, pp 267-275.

Andrews Z., Fitzgerald J. S., Payne R., Romanovsky A. 2013. Fault modelling for systems of systems, *In 11th Int'l Symposium on Autonomous Decentralized Systems, ISADS 2013:1-8,* Mexico City, Mexico IEEE.

Arnold A., Griffault A., Point G. and Rauzy A. 2000. The AltaRica formalism for describing concurrent systems. *Fundamenta Informaticae*, 40 (2-3):109-124, Amsterdam, IOS Press.

Azevedo L.S., Parker D., Walker M., Papadopoulos Y., Araujo R. 2013. Automatic Decomposition of Safety Integrity Levels: Optimization by Tabu Search. *2nd Workshop on Critical Automotive applications: Robustness & Safety (CARS) of the 32nd Int'l Conf. On Computer Safety, Reliability and Security (SAFECOMP),* September, Toulouse.

Azevedo L.S., Parker D., Walker M., Papadopoulos Y., Araujo R. 2014. Assisted Assignment of Automotive Safety Requirements. *IEEE Software* 31(1):62-68, Jan-Feb. 2014, IEEE.

Azevedo L.S., Parker D., Papadopoulos Y., Walker M., Sorokos I., Araujo R. 2014. Exploring the impact of different cost heuristics in the allocation of safety integrity levels. *Model-Based Safety and Assessment*: 70-81, Springer.

Bieber P., Delmas R., Seguin C. 2011. DALculus: Theory and Tool for Development Assurance Level Allocation, *Lecture Notes in Computer Science* 6894:43-56, Springer.

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* March 71(3):249–260.

Bozzano M. and Villafiorita A. 2007. The FSAP/NuSMV-SA Safety Analysis Platform. *International Journal on Software Tools for Technology Transfer*, February 9(1):5-24, Springer.

Chen D., Johansson R., Lönn H., Blom H., Walker M., Papadopoulos Y., Torchiaro S., Tagliabo F., Sandberg A. 2011. Integrated Fault Modelling for Safety-Critical Automotive Embedded Systems, *E&I Elektrotechnik und Informationstechnik*, June 128(6):196-202, Springer.

Dhouibi M.S., Perquis J. M., Saintis L., Barreau M. 2014. Automatic Decomposition and Allocation of Safety Integrity Level Using System of Linear Equations. *4th Int'l Conf. on Performance, Safety and Robustness in Complex Systems and Applications*, *PESARO* February, France, :1-5.

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

Eclipse Foundation (2016) Papyrus modelling environment. Available online at: http://www.eclipse.org/papyrus/

Edifor E., Walker M. Gordon N. 2012. Quantification of Priority-OR Gates in Temporal Fault Trees, *SAFECOMP'12, Computer Safety, Reliability and Security: Lecture Notes in Computer Science* 7612:99-110, Springer.

Edifor E., Walker M. Gordon N. 2013. Quantification of Simultaneous-AND Gates in Temporal Fault Trees. DepCos-RELCOMEX'13, *Advances in Intelligent Systems and Computing* 224:141-151, Springer.

Feiler P. H. and Rugina A.E. 2007. Dependability Modelling with the Architecture Analysis and Design Language (AADL). *Technical report, CMU/SEI-2007-TN-04*.

Fenelon, P., & McDermid, J. 1993. An integrated toolset for software safety analysis. *The Journal of Systems and Software* 21(3):279-290, Elsevier.

Grunske L., Kaiser B. and Papadopoulos Y. (2005) Model-driven Safety Evaluation with State-Event-Based Component Failure Annotations. CBSE'05, *Lecture Notes in Computer Science* 3489:33-48, Springer.

Int'l Electrotechnical Comission (2010) IEC 61508: Functional safety of electrical/electronic/programmable electronic safety related systems, 2nd edition. Geneva: IEC

Int'l Organization for Standardization (2011) ISO 26262: Road vehicles - functional safety. Geneva: ISO.

ITI GmbH. (2016) SimulationX Safety Designer. Available online at: https://www.simulationx.com/simulation-software/beginners/safety-designer.html

Joshi A., Vestal S., Binns P. 2007. Automatic Generation of Static Fault Trees from AADL Models. *DSN'07 Workshop on Architecting Dependable Systems*.

Jouault, F., Allilaire, F., Bezivin, J. & Kurtev, I., 2008. ATL: A model Transformation tool. *Science of Computer Programming* , June, 72(1-2):31-39.

Kabir S., Walker M., Papadopoulos Y. 2015. Quantitative evaluation of Pandora Temporal Fault Trees via Petri Nets. *9th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, Paris, France, 48(21):458-463.

Kabir S., Walker M., Papadopoulos Y. 2014a. Reliability Analysis of Dynamic Systems by Translating Temporal Fault Trees into Bayesian Networks, IMBSA'14, *Lecture Notes in Computer Science* 8822:96-109.

Kabir S., Edifor E., Walker M., Gordon N. 2014b. Quantification of Temporal Fault Trees Based on Fuzzy Set Theory. DepCos-RELCOMEX'14, *Advances in Intelligent Systems and Computing* 286:255-264, Springer

Kaiser, B., Liggesmeyer, P., & Mäckel, O. 2003. A new component concept for fault trees. *8th Australian Workshop on Safety Critical Systems*, 33:37-46.

Konak, A., Coit, D.W., Smith, A.E. 2007. Multi-objective optimization using genetic algorithms. *Reliability Engineering & System Safety*, September 91(9):992-1007.

Koziolek A., Koziolek H., Reussner R. 2011. PerOpteryx: automated application of tactics in multi-objective software architecture optimization. *ACM SIGSOFT symposium on Quality of Software Architectures, ISARCS*: 33-42, ACM.

Kwiatkowska M., Norman G., Parker D. 2009. PRISM: probabilistic model checking for performance and reliability analysis. *SIGMETRICS Performance Evaluation Review* 36(4): 40-45. ACM.

Mader R., Armengaud E., Leitner A., Steger C. 2012. Automatic and optimal allocation of safety integrity levels, *Reliability and Maintainability Symposium, RAMS*: 1-6, IEEE.

Mahmud N., Walker M., Papadopoulos Y. 2012, Compositional synthesis of Temporal Fault Trees from State Machines. *ACM SiGMETRICS Performance Evaluation Review*, 39(4):79-88.

Marsan, M.A., Balbo, G., Conte, G., Donatelli, S. and Franceschinis, G. 1996. *Modeling With Generalized Stochastic Petri Nets*. West Sussex: Wiley.

Mathworks (2016) Matlab Simulink. Available online at: www.mathworks.com/products/simulink/

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

Mian Z., Bottaci L., Papadopoulos Y., Sharvia S., Mahmud N. 2014. Model Transformation for Multi-objective Architecture Optimisation of Dependable Systems. *Advances in Intelligent Systems and Computing* 307:91-110, Springer.

Murashkin, A., Azevedo, L.S., Guo, J., Zulkoski, E., Liang, J.H., Czarnecki, K. & Parker, D. (2015) Automated decomposition and allocation of automotive safety integrity levels using exact solvers. SAE International Journal of Passenger Cars - Electronic and Electrical Systems, 8(1), 14 April.

[Accessed 30/4/2015].

Nggada, S. H., Papadopoulos, Y., Parker, D. J. 2013. Combined Optimisation of System Architecture and Maintenance. *IFAC DCDS'13*: 4:1 (25-30).

Ortmeier F., Güdemann M., Lipaczewski M.. 2012. Unifying Probabilistic and Traditional Formal Model Based Analysis. *8th Dagstuhl-Workshop on MBEES*: 123-132.

Papadopoulos Y., McDermid J. A., 1999. Hierarchically Performed Hazard Origin and Propagation Studies, *Lecture Notes in Computer Science* 1698:139-152.

Papadopoulos Y., Grante C. 2003. Techniques and tools for automated safety analysis & decision support for redundancy allocation in automotive systems, *27th Int'l Conf. on Computer Software and Applications, COMPSAC'03*: 105-110.

Papadopoulos Y., Walker M., Reiser M.-O., Weber M., Chen D., Törngren, Servat D., Abele A., Stappert F., Lönn H., Berntsson L., Johansson R., Tagliabo F., Torchiaro S., Sandberg A. 2010. Automatic Allocation of Safety Integrity Levels, *CARS'10*: 7-10, ACM.

Papadopoulos Y., Walker M., Parker D., Rüde E., Hamann R., Uhlig A., Grätz U., Lien R. 2011. Engineering Failure Analysis & Design Optimisation with HiP-HOPS, *Journal of Engineering Failure Analysis*, 18 (2): 590-608, Elsevier Science.

Parker D., Walker M., Azevedo L., Papadopoulos Y., Araujo R. 2013. Automatic Decomposition and Allocation of Safety Integrity Levels using a Penalty-based Genetic Algorithm., *Lecture Notes in Computer Science* 7906:449-459. Springer.

Sharvia S., Papadopoulos Y. 2008. Non-coherent Modelling in Compositional Fault Tree Analysis. In *Proceedings of the 17th International Federation of Automatic Control,* IFAC World Congress, Seoul, South Korea. Jul 2008.

Sharvia, S. et al., 2014. Enhancing the EAST-ADL Error Model with HiP-HOPS Semantics. *Athens Journal of Technology Engineering,* June 119-136

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

Society of Automotive Engineers (2010) ARP4754-A: Guidelines for development of civil aircraft and systems. Warrendale: SAE International.

Sorokos I., Papadopoulos Y., Azevedo L., Parker D., Walker M., 2015. Automating Allocation of Development Assurance Levels: an extension to HiP-HOPS. *IFAC Dependable Control of Discrete Systems*, 48(7):9-14.

Suresh P., Babar A., Raj V. 1996. Uncertainty in fault tree analysis: A fuzzy approach. *Fuzzy Sets and Systems,* October 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), pp. 453-457.

Vesely, W., Stamatelatos, M., Dugan, J., Fragola, J., Minarick, J. and Railsback, J. 2002. *Fault tree handbook with aerospace applications*. NASA office of safety and mission assurance, Washington DC, USA.

Walker M., Bottaci L., Papadopoulos Y. 2007. Compositional Temporal Safety Analysis. *Lecture Notes in Computer Science* 4680:105-119, Springer.

Walker M., Reiser M-O., Tucci S., Papadopoulos Y., Lonn H., Parker D., Chen D.-J. 2013. Automatic Optimisation of System Architectures using EAST-ADL *Journal of Systems & Software*, October 86(10):2467–2487.

Wallace M. 2005. Modular architectural representation and analysis of fault propagation and transformation. *Electronic Notes Theoretical Computer Science,* 141(3):53–71.

Wolforth I., Walker M., Grunske L., Papadopoulos Y. 2010. Generalisable Safety Annotations for Specification of Failure Patterns, *Software Practice and Experience,* April 40(5):453-483.

Yang L.2012. Analysis on Dynamic Fault Tree Based on Fuzzy Set. *Applied Mechanics and Materials,* 110-116:2416-2420.

Zeng W., Papadopoulos Y., Parker D. (2007), Reliability Optimization of Series-Parallel Systems Using Asynchronous Heterogeneous Hierarchical Parallel Genetic Algorithm. *Journal of Mind and Computation*, 1(4):403-412, China Academic Electronic Publishing House.

Zhang H., Li W., Qin J. 2010. Model-based Functional Safety Analysis Method for Automotive Embedded System Application. *Int'l Conf. on Intelligent Control and Information Processing*: IEEE 761-765.