IFAC-DCDS Plenary Paper, IFAC-PapersOnLine, Volume 48, Issue 7, 2015, Pages 1-8

A Synthesis of Logic and Biology in the Design of Dependable Systems

Yiannis Papadopoulos

University of Hull, Hull, HU6 7RX, UK
(+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 combines effectively and throughout the design lifecycle these two techniques which are schematically founded on the two pillars of formal logic and biology. 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, maintenability 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 However, these techniques are manual processes and often performed on informal system models which may rapidly become out of date as the system design evolves.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 as well as 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), and SAML (Ortmeier *et al.*, 2012).

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

In this paper, firstly, I discuss a set of challenges that in my view cannot be addressed by MBSA in its current state. These challenges mainly refer to design problems where there are potentially many design options to be considered. Secondly, I argue that a synthesis of these techniques with modern metaheuristics for search and optimisation can potentially address these challenges. Finally, I describe my work towards this goal within the HiP-HOPS method and tool, and 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, I discuss challenges; in Section 3, I present an extension of MBSA with metaheuristics; in Section 4, I discuss related work and, in Section 5, I conclude 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 it is a static field of research. It is rather a set of techniques which are evolving to address current and new challenges. I identify four such challenges which MBSA techniques cannot fully address.

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. 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 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 can integrity requirements 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 upon 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, 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. Not only does

the impact of these changes need to be localised, but any design change and reallocation of requirements that follow must be done efficiently so that the temporary imbalance caused by the design iteration is addressed with minimum disruption in the contractual relationships between 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 reconfigurations options at runtime. This creates difficulties in design because, as potential design spaces expand, their exploration for suitable or optimal designs becomes increasingly more 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 the various formulations of the above represent hard, multi-objective optimisation problems that can only be approached systematically with the aid of optimisation algorithms that can search in large potential design spaces.

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

3. SYNTHESIS OF MBSA WITH METAHEURISTICS

The above challenges go beyond the capabilities of current MBSA techniques. I believe that one step towards addressing them is by achieving 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, with my colleagues I have been working in this direction in the context of HiP-HOPS, an MBSA technique known which I have pioneered since the late 90s (Papadopoulos and McDermid, 1999). While HiP-HOPS has 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).

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).

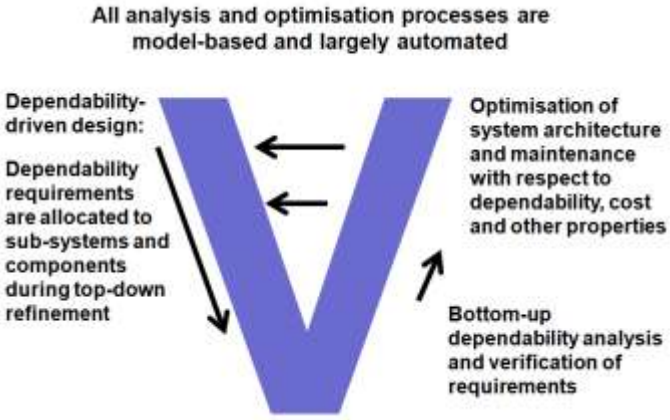

Fig. 1. Scope of HiP-HOPS in the V-lifecycle

At the early stages, HiP-HOPS supports a dependability driven mode of design in which system requirements captured early are allocated to sub-systems and components of the architecture as this is refined. 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 mainly defined 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).

As the architecture of the system is being refined and detailed models of the system are being produced with specific components selected and developed, detailed qualitative and quantitative dependability analysis can be done to produce evidence that 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).

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)

I will now outline the three steps in this process starting from modelling and dependability analysis in HiP-HOPS. This

capability is central to the method and provides the fundamentals that underpin the more advanced metaheuristic-driven processes of design refinement and optimisation.

3.2 Modelling and Dependability Analysis

All analysis and optimisation processes in HiP-HOPS are performed on a architectural system model which identifies material energy and data transactions among components (Fig. 2).

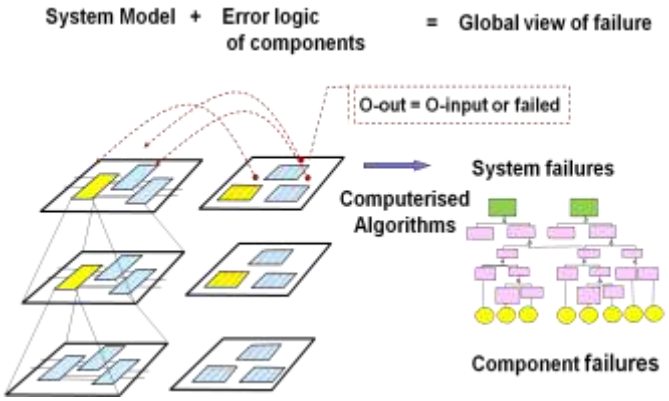

Fig. 2. Modelling and Dependability Analysis in HiP-HOPS.

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 first step in the analysis of such models is the annotation of each component in the model with its local error logic. 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.

Input and output errors referenced in the error logic are described qualitatively and typically represent conditions 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). 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.

We should note that experimental versions of the tool use an extended language where it is also possible to use temporal operators to describe sequencing of faults (Walker *et al.*, 2007) and wildcards to describe more abstract patterns of relationships between output and input deviations, for instance to allow statements such as "there will be an omission of all outputs in response to any input error" (Wolforth *et al.*, 2010).

Once the model has been augmented with error logic, then computerised algorithms are used to automatically determine how errors propagate through connections in the model and cause functional failures at system outputs. Those failures are the ones that analysts are typically interested to identify and analyse. For example, in a car, such functional failures would include the loss or incorrect steering and braking.

Via the automated analysis algorithms of HiP-HOPS, it is possible to move from a local component view of failure to a global system view of failure. This global view is captured in a set of fault trees which show how the component failure modes representing the leaf nodes of the trees and their local effects logically combine and propagate though the system to cause functional failures that represent the top events of the fault trees. Note that these trees are interconnected and share branches and leaf nodes which represent dependencies in model, e.g. the failure of a common power supply or a global condition that may affect more than one system functions. HiP-HOPS can perform qualitative cut-set analysis of fault trees. Quantitative analysis is also feasible 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 the failure mode on the system. The table shows whether, and how, each failure mode causes one or more system failures (i.e. top events of fault trees) by itself or 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 functional effects that a particular component failure mode causes. The latter 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. Precisely because it records the effects of combinations of component failures, this type of FMEA can, in practice, help analysts not only to locate problems in the design, but also to determine the level of fault tolerance in the system, i.e. to determine whether the system can tolerate any single 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.

3.3 Architecture and Maintenance Optimisation

Let us assume now that a team of analysts is designing a system and that an MBSA tool suggests that the system does not meet all dependability requirements. How should the analysts proceed? Should they start a radical redesign or explore simpler options first? And is manual redesign the only option or can there be some automation in design improvements. There is typically a range of options that designers will try whilst improving a design, and these include: 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, and d) increasing the frequency of maintenance, an action that prolongs the useful life of components and thereby increases the reliability of the system.

However a difficulty encountered here is that, in a typical design, the options for substitution, replication and maintenance scheduling are too many. For instance, in a system of n components, if there are two suppliers for each component then there are 2^n potential different configurations of components. For n=100, there are $1.26e^{30}$ configurations, each representing a different system configuration with its own dependability and cost performance. It is clear that in such situations analysts are confronted with a range of multi-objective optimisation problems, where the objectives are many and include dependability, cost, weight and other properties.

To provide a solution to this range of problems we have developed an extension of HiP-HOPS that employs genetic algorithms to enable multi-objective optimisation of architectures with respect to dependability and other attributes (Papadopoulos and Grante, 2006) (Papadopoulos *et al.*, 2011). The concept is illustrated in Fig. 3.

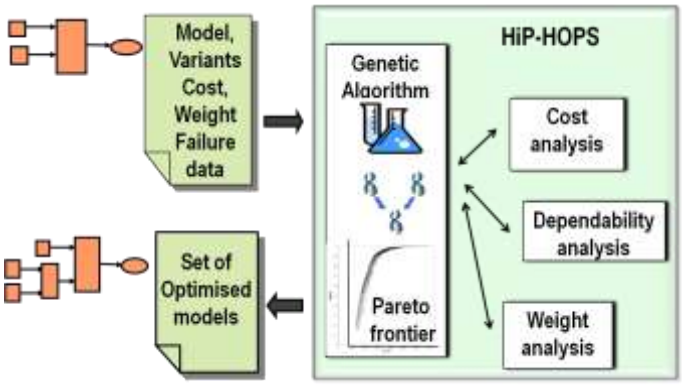

Fig. 3. Architecture optimisation in HiP-HOPS.

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. Components can have multiple alternative implementations. For example, a sensor can have two options from two different suppliers, with each option having its own cost, weight 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 architecture. 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.

Once the system model has included these possibilities, the model with its variants, failure data and cost data is given to HiP-HOPS which 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 selecting particular design options. Each candidate design is then evaluated with respect to the objectives of the optimisation. The process is aided by the automated algorithms of HiP-HOPS. The reliability or availability of a candidate design are calculated from fault trees. A quantitative measure of safety is established from the FMEA, for instance from the number of single points of failures that contribute to severe system failures. HiP-HOPS includes simple summative cost and weight functions. However, external plugins can also be designed to enable more precise evaluation of cost, weight or other objective functions. Experiments with timing and schedulability have been reported in (Walker *et al.*, 2013).

Once candidate designs have been evaluated, they are ranked according to performance, and a Pareto frontier is formed showing the best designs in the current population. Roulette selection that favours better performing designs is used to select candidates that 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 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 tradeoffs 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 making sure that dependability requirements can be met whilst minimising costs.

3.4 Dependability-driven Design Refinement with HiP-HOPS

The processes described so far are bottom up, i.e. they rely on the existence of detailed system models that can be subjected to analysis or optimisation. At this point, I would like to pose a set of more fundamental questions about design.

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

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

c) Why not imagine 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 standards. Indeed many standards express the desire for a top-down design process driven by requirements. However, the guidance provided by these standards is based on manual processes of requirements allocation. I 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, ISO 26262, and ARP4754-A, 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, and 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 process as well as the bottom-up verification of those requirements. For example, 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 what hazards may arise as a result. The severity, likelihood, and controllability of these hazards are then considered, and 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 hazard, the higher the ASIL that gets applied.

During 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. Then, during the verification and validation, analyses (e.g. fault trees) must be produced to ensure that the refined system meets original requirements.

However, the process of allocating and decomposing ASILs across the architecture is far from straightforward. The integrity levels of elements should be derived in such a way that the integrity level of their parent system is maintained, and thus in general the sub-components inherit the integrity level of their parent. To avoid every element of the entire system having to meet the highest level of integrity, fault tolerant architectures (e.g. parallelism, redundancy, monitoring etc.) can be employed to spread the burden of meeting the 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 ASILs (B, B) or (C, A) or (A, C) or (D, QM) or (QM, D). In practical design these options multiply and the question arises which design option is cost-effective. In general, an element of the system architecture should be allocated a SIL that corresponds to the SIL of the most severe failure it causes at system level. In simple systems, it may be easy to establish this fact. However, the correct, cost-optimal allocation of SILs in complex, networked architectures delivering multiple functions is far from trivial. Problems may include ensuring that independence constraints are met, whilst working through the many possible allocations, and ensuring that the decomposed low-level requirements still add up to the original high-level requirements. The above add to the time, expense, and complexity of following safety standards like ISO 26262 and ARP4754-A when dealing with the types of detailed electronic architectures that are common in modern safety-critical systems, often to the extent that performing the process manually is practically impossible.

In HiP-HOPS the above process has been largely automated by using the framework presented in section 3.2. HiP-HOPS can establish the potential contribution of assumed component faults and combinations of such faults to system level failures. From this information, it is then possible to automatically do a rational allocation and decomposition of requirements. HiP-HOPS is aware of which components are dependent by means of the propagation model and the fault tree analysis, and it can automatically apply the type of SIL 'algebra' described by e.g. ISO 26262 and ARP4754-A. The result is an allocation of SILs to individual components, component ports, or even component failure modes, in such a way that the overall high-level requirements are still fulfilled.

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 algorithms for a cost-optimal allocation can been problematic (Parker *et al.*, 2013). Therefore, recent work in HiP-HOPS has focused on the use of metaheuristics in order to efficiently explore this large space. This allows the generation of optimal or near optimal allocations of SILs that meet the system requirements with minimal costs for procurement, development and verification of components. Work in this area has included the use of genetic algorithms (Parker *et al.*, 2013) and Tabu search (Azevedo *et al.*, 2014), while more recently we are investigating particle swarm-based approaches. All this effort has produced promising results that are far more efficient and scalable than deterministic solutions, and a prototype implementation has been incorporated into the HiP-HOPS.

HiP-HOPS now calculates optimal or near 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 contracts 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 at the end of the design refinement process if the contracts have been met and basic components of the architecture meet their assigned requirements; assuming of course that any assumptions of independence made in the model have not been violated.

It is worth noting that 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 operate on adjustable rules and algebras which makes it compatible with a variety of contemporary standards. Compatibility has already been demonstrated with the ISO2626 automotive (Azevedo *et al*., 2014) and the ARP4754-A aerospace standards (Sorokos *et al*,, 2015).

4. 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 were then optimised using meta-heuristics. HiP-HOPS supersedes this work by enabling optimization of models which may have networked architecture, i.e. they are not necessarily in parallel/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 like 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 been the first effort 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 optimization problem is formulated to discover a solution that minimizes 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 formalize the ARP 4754a DAL allocation rules and the DALculator tool to support automatic DAL allocation. The starting point for these approaches are minimal cutsets of fault trees. HiP-HOPS starts from architectural models, offering the advantage of being able to assess dependencies in the model.

5. CONCLUSIONS

The technologies of model-based design, dependability analysis, and the application of heuristics in 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 refinement of new designs.

I 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 which 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 components selection, fault tolerance and maintenance scheduling; effective reuse of repositories of models and analyses both during design refinement and across projects. I have sketched this model-based design paradigm and outlined potential benefits. I have also shown that this is a feasible idea 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 from 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).

There are of course many 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, the scalability of models and computationally expensive analyses, and obstacles in the practicability and uptake of this work.

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

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

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

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.

Bozzano M. and Villafiorita A. 2007. The FSAP/NuSMV-SA Safety Analysis Platform. *Journal of Software Tools for Technology Transfer*, 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, *IE&I elektrotechnik und informationstechnik*, 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*:1-5.

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.

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

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*, 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.

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.

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*, 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.

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.

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

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*, 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*, 40(5):453-483.

Zeng W., Papadopoulos Y., Parker D. (2007), Reliability Optimization of Series-Parallel Systems Using Asynchronous Heterogeneous Hierarchical Parallel Genetic Algorithm, *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 Applications, *Int'l Conf. on Intelligent Control and Information Processing*: 761-765.