Engineering Failure Analysis & Design Optimisation with HiP-HOPS

Yiannis Papadopoulos [a]*, Martin Walker [a], David Parker [a], Erich Rüde [b], Rainer Hamann [b], Andreas Uhlig [c], Uwe Grätz [c], Rune Lien [d]

*Corresponding author: Y.I.Papadopoulos@hull.ac.uk, +44 (0)1482 465981
[a]University of Hull, Cottingham Road, Hull, HU6 7RX. United Kingdom
[b]Germanischer Lloyd AG, Brooktorkai 18, 20457 Hamburg, Germany
[c]ITI GmbH, Webergasse 1, D-01067 Dresden. Germany
[d]Agito SA, Postbox 792, N-3606 Kongsberg, Norway

Abstract
The scale and complexity of computer-based safety critical systems, like those used in the transport and manufacturing industries, pose significant challenges for failure analysis. Over the last decade, research has focused on automating this task. In one approach, predictive models of system failure are constructed from the topology of the system and local component failure models using a process of composition. An alternative approach employs model-checking of state automata to study the effects of failure and verify system safety properties.

In this paper, we discuss these two approaches to failure analysis. We then focus on Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) - one of the more advanced compositional approaches - and discuss its capabilities for automatic synthesis of fault trees, combinatorial Failure Modes and Effects Analyses, and reliability versus cost optimisation of systems via application of automatic model transformations.

We summarise these contributions and demonstrate the application of HiP-HOPS on a simplified fuel oil system for a ship engine. In light of this example, we discuss strengths and limitations of the method in relation to other state-of-the-art techniques. In particular, because HiP-HOPS is deductive in nature, relating system failures back to their causes, it is less prone to combinatorial explosion and can more readily be iterated. For this reason, it enables exhaustive assessment of combinations of failures and design optimisation using computationally greedy meta-heuristics.

1 Introduction

Increasing complexity in the design of modern engineering systems challenges the applicability of rule-based design and classical safety and reliability analysis techniques. As new technologies introduce complex failure modes, classical manual analysis of systems becomes increasingly difficult and error prone.

To address these difficulties, we have developed a computerised tool called 'HiP-HOPS' (Hierarchically Performed Hazard Origin & Propagation Studies) that simplifies aspects of the engineering and analysis process. The central capability of this tool is the automatic synthesis of Fault Trees and Failure Modes and Effects Analyses (FMEAs) by interpreting reusable specifications of component failure in the context of a system model. The analysis is largely automated, requiring only the initial component failure data to be provided, therefore reducing the manual effort required to examine safety; at the same time, the underlying algorithms can scale up to analyse complex systems relatively quickly, enabling the analysis of systems that would otherwise require partial or fragmented manual analyses.

More recently, we have extended the above concept to solve a design optimisation problem: reliability versus cost optimisation via selection and replication of components and alternative subsystem architectures. HiP-HOPS employs genetic algorithms to evolve initial non-optimal designs into new designs that better achieve reliability requirements with minimal cost. By selecting different component implementations with different reliability and cost characteristics, or substituting alternative subsystem architectures with more robust patterns of failure behaviour, many solutions from a large design space can be explored and evaluated quickly.

Our hope is that these capabilities, used in conjunction with computer-aided design and modelling tools, allow HiP-HOPS to facilitate the useful integration of a largely automated and simplified form of safety and reliability analysis in the context of an improved design process. This in turn will, we hope,

address the broader issue of how to make safety a more controlled facet of the design so as to enable early detection of potential hazards and to direct the design of preventative measures. The utilization of the approach and tools has been shown to be beneficial in cases studies on engineering systems in the shipping [1] and offshore industries [2].

This paper outlines these safety analysis and reliability optimisation technologies and their application in an advanced and largely automated engineering process. In section 2 we discuss the context behind these technologies and discuss some related approaches. In section 3, we describe how automatic safety analysis can be performed using HiP-HOPS, and then in section 4 we relate the new optimisation capabilities of HiP-HOPS. We apply these capabilities to a simplified marine fuel oil system in section 5, and highlight the benefits of using automated tools in this way. Finally in section 6 we discuss these recent developments in HiP-HOPS and locate them in the context of other contemporary research in the area, and then in section 7 we present our conclusions.

The paper makes two contributions: firstly, it gives a summary of the current state of HiP-HOPS, focusing on recent work on optimisation, and relates this approach to the present state of the art; secondly, it presents a case study on optimisation of system architecture using genetic algorithms and shows how this technique can automate, rationalise and simplify the classical cost benefit analysis that drives decisions about system optimisation. The example provides an opportunity to discuss strengths and limitations of HiP-HOPS in relation to other state-of-the-art techniques. We show that HiP-HOPS is less prone to combinatorial explosion and can therefore be used for design optimisation using computationally greedy meta-heuristics.

2 Safety Analysis & Reliability Optimisation

2. 1 Classical Safety Analysis Techniques: FTA & FMEA

Fault Tree Analysis (FTA) [3] and FMEA [4] are well-known and widely used system analysis techniques used in reliability engineering. Both are long established – FMEA was formally introduced in the late 1940s, and FTA has been around since the 1960s – and both have been employed in a number of different areas, including the aerospace, nuclear power, and automotive industries. They are methods that we can use to identify potential faults in a system, so that we can then use that information to correct or prevent those faults.

Fault Tree Analysis (FTA) is a flexible technique, equally applicable to quantitative and qualitative analyses, and easy to use and understand. Fault trees themselves are graphical representations of logical combinations of failures, and show the relationship between a failure or fault and the events that cause them. A fault tree normally consists of a *top event*, which is typically a system failure, connected to one or more *basic events* via a system of logical gates, such as AND and OR. Basic events are usually either component failures or events expected to happen as part of the normal operation of the system. Analysis of the fault tree consists of two parts: *qualitative* (logical) analysis, and *quantitative* (probabilistic) analysis. Qualitative analysis is performed by reducing the logical expression represented by the fault tree into a set of *minimal cut sets*, which are the smallest possible combinations of failures required to cause the top event. Quantitative analysis is performed by calculating the probability of the top event given the probability of each of the basic events occurring.

In an FMEA, the basic process consists of compiling lists of possible component *failure modes* (all the ways in which an entity may fail), gathered from descriptions of each part of the system, and then trying to infer the effects of those failures on the rest of the system. Usually, these effects are evaluated according to a number of criteria, such as severity, probability, and detectability, and often these criteria are then combined into an overall estimate of risk. All of this data is then presented in the form of a table which allows the analyst to quickly see what the effects of each failure mode are.

There are obvious differences between the two techniques, but one of the most fundamental is the direction of the analysis. FTA is a deductive technique, which means it works from the *top down* – assuming the system has failed, and then trying to work out why it failed. This is done by working backwards to determine what possible combinations of events might have caused it; the system failure then becomes the top event of the fault tree and the individual component failures form the basic events, and they are all combined using a network of logical gates. FMEA, by contrast, is an inductive technique, and works from the *bottom up* – assuming a component failure has occurred, and then trying to work out what its effects would be. It involves proposing a certain event or condition, and then trying to assess the effects of that initial event on the rest of the system. The end result is a table of

failures and their effects on the system, which provide the analyst with an overview of the possible faults.

Both techniques are useful and provide a lot of valuable information about systems, and each can be used to complement the other, but both suffer from the same flaw: they are primarily manual methods. The process of performing these analyses can be laborious, especially for larger and more complex systems. Whilst this provides the analyst with an in-depth knowledge of the system being studied, it also makes application of these techniques error prone, and the results (once obtained) are often too numerous to interpret efficiently. As a consequence, it is not uncommon for FTA and FMEA to take place only once or twice in the life cycle of the system. This is unfortunate, because systems analysis techniques like FTA and FMEA can be of great benefit during an iterative design process. By estimating the reliability and gaining a more thorough understanding of the failure behaviour of the system in each iteration, it is possible to see how the changes in design impact upon the overall safety of the system. It also enables the analysts to identify and remedy potential flaws much earlier, thereby saving both time and effort and producing a more reliable product.

However, before FTA or FMEA can be incorporated into the design cycle in this way, it is necessary to overcome the difficulties inherent in such manual techniques. Experience from the aerospace and process industries suggests that the application of classical safety analysis is hindered by the increasing complexity of systems. For relatively simple systems, this is a manageable process, although fault trees and FMEAs can rapidly become very elaborate. In complex systems, however, manual analysis is laborious and error prone, and a thorough assessment and interpretation of the results becomes increasingly difficult to achieve within the constraints of most projects. Furthermore, the results of the analyses are divorced from the design being analysed, meaning that the effects of any changes in the system design may only become apparent after another long and costly analysis.

While guidance is available on how technical justification for alternative designs and arrangements can be provided in the form of engineering and safety analyses, such as FTA and FMEA, there is a need for specific supporting measures and tools to assist in the application of such techniques. One obvious approach would be to automate at least part of the process. This would mean that the analyses could be carried out more quickly and efficiently, leaving more time for the results to be studied and allowing more useful conclusions to be drawn.

2.2 Modern Safety Analysis Techniques & Tools

The deficiencies in manual safety analysis techniques like FTA and FMEA have not gone unrecognised, and over the years a number of different tools and techniques have been established to try to automate the process. Although pure analysis tools have a long history, e.g. FTA tools range from modern examples like Fault Tree + [5] to more venerable tools like SETS [6], these typically require the fault tree (or FMEA etc) to be built manually. Recently, however, tools have begun to integrate more closely with the design process, allowing the synthesis of the fault models (i.e. fault trees, FMEA) to be achieved automatically. This enables designers and safety analysts to better take advantage of ever-increasing computer processing power and also helps to facilitate the reuse of information as part of a more iterative design process in which safety becomes a driving factor.

Many of these tools and techniques have tended to fall into one of two categories [7]. The first category consists of the compositional safety analysis approaches: the development of formal and semi-formal languages to enable specification, composition, and analysis of system failure behaviour based on safety information about the components that comprise the system. Although most are not fully automated, in that they depend on the manual entry of the initial component failure data, they can be usefully applied in the analysis of both hardware and software architectures in the context of a model-based design process. The second category centres on adaptations of formal verification techniques to support safety analysis. These tend to involve more rigorous modelling to enable model-checking and related processes to take place, analysing the effects of failures by simulating them and verifying whether or not the system meets its safety goals in the presence of those failures.

2.2.1 Compositional Safety Analysis Techniques

Perhaps the earliest compositional safety analysis technique, and one that has influenced those that followed, is the Failure Propagation and Transformation Notation (FPTN), which emerged in the 1990s [8]. It is a graphical description of the failure behaviour of the system, based around the idea of component modules which describe the generation and propagation of component failures in the

system. These component modules are connected via inputs and outputs to other modules, allowing combination and propagation of failures from one module to another, and they can be composed into subsystems that can be used to build a system hierarchy. FPTN was designed to provide a bridge between the deductive FTA and inductive FMEA processes, allowing both cause and effect to be studied. However, FPTN's component module approach means building an error model that is separate from the system model, which is then prone to becoming desynchronised from the original system it represents as the design evolves [9]. As such, the original FPTN remained primarily a notation for describing specifications of failure. Unlike HiP-HOPS, it was never extended with capabilities for system analysis or design optimisation.

The concepts introduced in FPTN were taken further in the Fault Propagation and Transformation Calculus (FPTC) [9], which was an attempt to overcome the deficiencies in FPTN. The main way in which this was achieved was to link the failure model to the architectural model so that all dependencies are identified and maintained. FPTC defines different failure classes (like omission, commission, value errors etc) and these are specified in annotations directly in the components of the system model. The inputs and outputs of those components are then used to transmit failure information to the rest of the system by using a set of expressions that detail how failures are transformed and propagated from input to output. Representing mitigation of failure is possible by transforming a failure into normal behaviour. The failure propagation defined by these expressions and the original model structure can then be evaluated as a token-passing network, determining the effects of each component failure on the system as a whole. FPTC has also been extended [10] to permit quantitative (probabilistic) analysis by including probability values in each expression. The major advantage of FPTC over FPTN is that it uses the actual architectural model of the system, meaning that any localised changes to the model do not require a new failure model to be built; instead, only a subset of the failure expressions need updating. However, whereas FPTN was designed to support both inductive and deductive analysis, FPTC is primarily inductive in nature, since it relies upon injecting one failure or combination of failures into the system at a time and repeating the analysis. As a result, the type of information given by an FTA is more difficult to achieve with FPTC and its inductive nature makes it prone to combinatorial explosion, particularly for systems with large numbers of failure modes.

Other approaches build upon the foundations laid by FPTN in other ways. Two such examples are State-Event Fault Trees (SEFTs) [11, 12] and Component Fault Trees (CFTs) [13, 14]. In the CFT approach, the failure logic of components is defined as a graph of interconnected fault trees using a specification similar to that used in FPTN and HiP-HOPS. The CFTs themselves can be composed to produce larger CFTs for subsystems to build the system hierarchy. Because CFTs are based on fault trees, they are less prone to the problems with combinatorial explosion that afflicts FPTC. SEFTs are a development of CFTs designed to overcome the static logic inherent in FTA and FMEA and enable analysis of dynamic systems that feature transitions between different states. Whereas most FTA/FMEA-based approaches feature only basic failure events, SEFTs distinguish between a system being in a certain state (which is a condition that is true over a period of time) and an event that triggers a state transition (which is an instantaneous occurrence); this makes SEFTs better suited to analysing software systems or hardware systems with more complex dynamic behaviour. As with FPTC, failure behaviour is modelled at the component level, but the simple Boolean logic of FPTC is extended to enable the representation of sequences and histories of events, as well as the concept of negation (i.e. an event that has not happened yet) using the NOT gate. However, this more complex logic means that analysis of SEFTs is not possible using traditional FTA algorithms and instead relies upon a conversion to Deterministic Stochastic Petri Nets (DSPNs) [15]. These DSPNs can then be quantitatively analysed using Petri Net tools like TimeNET [16]. The disadvantage of modelling different states is that the state-space can grow exponentially in larger models, reducing the scalability of the SEFT approach.

Finally, HiP-HOPS also falls into this category, since it is based upon the synthesis and analysis of fault trees using compositional component failure data. However, it will be described in detail in section 3 and its capabilities will be discussed and compared with other approaches in section 6.

2.2.2 Extensions of formal verification techniques for safety analysis

Another category of modern safety analysis techniques involves the use of formal modelling engines to simulate the normal functioning of the system design and then inject faults to determine their effects on the system. The benefit of this type of approach is that it does not typically require any additional annotation (e.g. to describe the propagation of failures and component failure logic) as all required

information can be automatically extracted from the model, although this is only possible where simulation data for the domain in question is available in the first place.

There are a number of different tools and techniques that fall into this category. Two of the best known are FSAP-NuSMV [17] and Altarica [18]. Formal Safety Analysis Platform (FSAP) is a graphical interface for the NuSMV-SA model checking & safety analysis engine; together they provide a single environment for model design and safety analysis. It is capable of both injected fault simulation and normal model checking processes like property verification and counter-example generation. FSAP-NuSMV can be used to generate fault trees, although these fault trees essentially consist only of top events and basic events and therefore link directly from effects to combinations of causes, although they can also include NOT gates. Its main drawback, in common with the other model-checking approaches, is its susceptibility to combinatorial explosion as the number of potential failure modes increases.

Altarica is a formal modelling language that can describe complex hierarchical models [19]; it can also model states and events and is therefore capable of representing the behaviour of dynamic or state-based systems. As with SEFTs described above, Altarica can be used to generate Petri Nets as well as static fault trees for non-dynamic systems. In both cases, the failure model is then analysed by external tools. One issue with Altarica, as identified by Bieber in [18], is that it can be subject to loops of fault propagation logic in the model, particularly when bidirectional signals or flows are involved. In these cases, the SMV-based approach is usually employed instead [20].

There are other examples of this category, for example Model-based Deviation Analysis (MDA) [21] and Deductive Cause Consequence Analysis (DCCA) [22]. MDA differs from the others in that it is less concerned with system faults and instead attempts to model the effects of deviations of system input data. It is an extension of the earlier Software Deviation Analysis [23] that seeks to relate deviations of input data to deviations of output. MDA uses quantifiable limits and ranges that can be formulated in temporal logic and then simulated using a model checker. By comparing simulations of nominal data and error data, critical deviations from the normal can be detected and their causes identified. One downside of MDA, however, is its requirement for two datasets (nominal and error), although it is possible in some cases for these to be combined in the same model or environment.

DCCA is similar to the other model-checking approaches and uses mathematical proofs to verify whether a component failure will lead to system failure. The behaviour of the system is defined using finite automata and temporal logic; the system can then be simulated with regard to a set of component faults and system hazards to determine whether those component faults will cause one or more of the system hazards being investigated. The goal is to determine the minimal critical set, i.e. the smallest combination of failure modes capable of causing a hazard. Other model checkers like SMV can also be used to verify the proofs. DCCA offers some other advantages: it has been modified to work with an industrial design tool, SCADE [24], and although it also suffers to a degree from combinatorial and state-space explosion, this can be minimised by making prior use of more informal techniques like classical FTA to determine failures of interest to investigate further. DCCA also features an extension named DFOA (Deductive Failure Order Analysis) that enables analysis of dynamic systems using temporal fault trees generated automatically from the system model [25].

2.2.3 Implications

Although both kinds of approach have advantages and disadvantages [26], e.g. in terms of the level of automation, amount of detail provided, scalability to handle multiple failures in combination, or ability to analyse dynamic systems with multiple states, all of them offer a distinct advantage over classical manual techniques in that they can utilise tool support and therefore offer a much faster, more detailed analysis. This makes it more practical to use these safety analysis techniques as part of the design process, and in some cases multiple techniques can be used to complement each other.

However, if some portion of the design evolution is to be iterated automatically by using a model-based optimisation algorithm, then the compositional safety analysis approach offers one significant advantage over the formal approaches: performance. Although the compositional approaches require the prior annotation of the model with failure logic, which can be time consuming, subsequent synthesis and analysis of fault trees etc is typically very fast, since it normally operates in a deductive manner and is less prone to combinatorial explosion. By contrast, although formal fault simulation techniques like Altarica etc do not require this initial step, since the required information can be

automatically extracted from domain libraries of components with failure properties, the actual simulation and analysis is typically inductive and therefore slower and more vulnerable to combinatorial explosion. As a result, it is usually impractical to consider the effects of combinations of failures, or at least more than two failures in conjunction.

Automatic design optimisation, e.g. with metaheuristics such as genetic algorithms, is an iterative process that may require a model to be evaluated thousands or even millions of times. As a result, the performance and therefore scalability of the evaluation technique is a critical factor. Furthermore, since the evaluation is being used primarily to differentiate different potential design candidates, the level of detail that can be provided by fault simulation approaches is often unnecessary. Therefore, the compositional approaches tend to be better suited to this process; they can of course be complemented by an analysis of certain promising design candidates using a fault simulation technique once the optimisation has finished.

3 Automatic Optimisation of System Reliability

3.1 Need for automatic optimisation

As described above, modern safety analysis techniques permit a wealth of information about the safety and reliability of a system to be gathered much more quickly and easily than ever before. This information permits designers to use safety and reliability as a major factor in the decisions they make during the evolution of a system design: by evaluating the effects of one or more potential design choices, e.g. increased reliability at the expense of greater cost or increased weight etc, designers are able to make informed choices.

However, just as classical manual safety analyses restrict the rate that information can be obtained about a system, manually evaluating different design choices is time-consuming and restricts the number of design candidates that can be investigated. If this process could be automated, it would be possible to examine hundreds or thousands of potential design candidates – a much greater portion of the total design space – and thus hopefully provide a better foundation for the next iteration of the system design.

Unfortunately, even automation using modern computing technology is insufficient to examine the total design space even for a relatively simple system with a limited number of substitutable components. It is therefore not possible to evaluate *every* design candidate except in the most trivial of cases. This is particularly the case if multiple methods of variability are taken into account, e.g. swapping one subsystem architecture for another as well as just substituting one component for another with an alternative implementation.

Furthermore, an additional issue is that system reliability is seldom the only concern of the designer: the major constraint on any design is likely to be its cost, and as most characteristics of a system increase, cost typically increases too. These conflicting goals – e.g. to increase reliability while reducing cost – result in any design having to reach a careful balance between different objectives. Any automated optimisation process must therefore take into account not just a single objective, but multiple – and often contradictory – objectives.

3.2 Comparison of different optimisation approaches

There are a number of different optimisation algorithms available, most revolving around the use of meta-heuristics intended to explore the design search space as extensively and as effectively as possible; the aim is to find the optimal solutions as soon as possible but without getting trapped in one local area of the search space (known as a 'local optimum').

One such algorithm is known as *tabu search* [27, 28]. Tabu search explores the design space on the basis of evaluation functions, e.g. if one solution has better characteristics, it uses that one as the basis for its next iteration. Multiple objectives can be evaluated by multiple functions in multiple iterations, e.g. objective #1 in the first iteration, objective #2 in the second, and so forth. Furthermore, tabu search remembers the solutions it has explored, preventing the algorithm from looping or getting stuck in a local optimum; these are known as the taboo or tabu solutions and force the algorithm to focus on unexplored areas of the search space. The list is typically finite, however, and eventually the oldest members of the list are removed. After a set number of iterations, the best solution(s) found so far are presented as the results.

A different approach is the *ant colony* approach, which is based on the behaviour of ant colonies in nature [29, 30]. Ants use pheromone trails to establish and navigate efficient routes between food sources and their nest; poor routes disappear over time as less ants use them, while good routes are reinforced by more and more ants using them. Ant colony optimisation works in the same way: in each iteration, one solution is used as the base and the neighbouring solutions are explored by the 'ants'. Good solutions are reinforced and in the next iteration, the best solution found is used as the next base or 'nest'.

Simulated annealing is another technique, this time based on metallurgical annealing (i.e. controlling the rate of cooling in metal to encourage the atoms in the metal to form stronger shapes) [31]. In simulated annealing, the current solution is changed in some random way to obtain a different solution; whether or not that new solution is accepted depends on the global temperature: a hot temperature allows more radical changes, a cool temperature permits only minor changes. At each stage, the most optimal solution is chosen from those allowed, and then the temperature is lowered and the process continues.

Another approach is to use *genetic algorithms* [32]. Genetic algorithms are designed to mimic the evolution of biological life in nature: a population of different candidates are evaluated according to their fitness and the best are chosen to reproduce and form the basis of the next generation of candidates. Each candidate is represented using a 'genetic' encoding, which describes its characteristics; when two candidates are chosen to produce a new design candidate, their encodings are subjected to *crossover*, i.e. the encoding of the new design is a mix of the encodings of its ancestor candidates. This promotes convergence on optimal solutions by ensuring 'survival of the fittest'. In addition, random mutation is included to allow a greater portion of the search space to be explored and to try to limit the likelihood of becoming stuck in a local optimum.

There are different versions of genetic algorithms tailored towards different goals. One form of genetic algorithm is the penalty-based approach; in this form, the multiple objectives are combined into a single function. One objective (e.g. reliability) is optimised, but constraints on the other objectives (e.g. cost, weight etc) are imposed and any infringement of those constraints incurs a penalty which is subtracted from the fitness score for the candidate. Thus if two potential solutions have the same reliability, but one violates the constraints, it is ranked lower than the one that does not violate the constraints. The penalty incurred can be tweaked so that the design space is not too confined, e.g. by having smaller penalties at the start of the optimisation and steadily increasing them as time goes on, which can result in better solutions [33].

Another form of genetic algorithm is the Non-Dominated Sorting Genetic Algorithm (NSGA-II) [34], which is a true multi-objective approach (as opposed to the penalty-based approach, which combines multiple objectives into one). Full multi-objective approaches tend to be better at exploring the design space more widely, albeit typically with a performance penalty, since more evaluation is taking place. NSGA-II works by constructing a multi-dimensional graph plotting the current solutions in the population, as in the example of Figure 1.

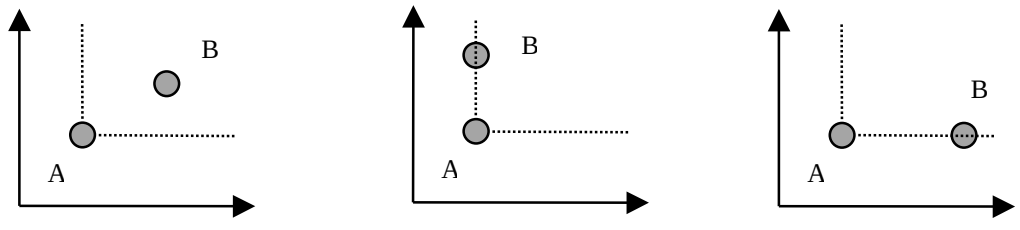


Figure 1: Dominated & non-dominated solutions

For a given solution B, if there is another solution A that is better in at least one objective and no worse in any others, then solution B is said to be *dominated* by A. The set of all non-dominated solutions is known as the *Pareto front*, and is the set of currently identified optimal solutions. These represent different trade-offs between the multiple objectives, e.g. one might have the best reliability for a given high cost, whereas another might have the lowest cost for a lower reliability value; at one extreme the

Pareto front would contain the cheapest solution and at the other extreme it would contain the most reliable. The nature of the Pareto front means that it is not possible to move from one solution to another without sacrificing at least one objective, e.g. one could not choose a more reliable solution from the Pareto front without also increasing cost (or some other objective). The Pareto front can be managed using different techniques to ensure a good base population for the next generation, e.g. by pruning out clusters of similar solutions (so the Pareto solutions remain more widely spread out over the search space).

3.3 Choosing an optimisation algorithm

Although there are many different optimisation techniques – and many variations of individual techniques – for the purposes of design optimisation, the most suitable techniques are those that allow multiple objectives to be considered, since safety and/or reliability are rarely the only factors being considered. In particular, cost is typically a critical factor, and depending on the nature of the system, other objectives like weight, volume, or timing (i.e. performance of software) may form elements of the optimisation as well.

For our purposes in HiP-HOPS, we developed a modification of the NSGA-II variant of the genetic algorithm approach. Genetic algorithms offer one of the best balances between design space exploration and convergence on optimal solutions because of the combination of the crossover and mutation operators. They are also readily adaptable to multiple objectives. Although different multi-objective genetic algorithms exist, e.g. PESA-II and SPEA2, NSGA-II is regarded as efficient and well-tested [35] and it is more open to modification than some of the others. The optimisation process used in HiP-HOPS is described in more detail in section 4.

4 Safety Analysis using HiP-HOPS

HiP-HOPS is a compositional safety analysis tool that takes a set of local component failure data, which describes how output failures of those components are generated from combinations of internal failure modes and deviations received at the components' inputs, and then synthesises fault trees that reflect the propagation of failures throughout the whole system. From those fault trees, it can generate both qualitative and quantitative results as well as a multiple failure mode FMEA [36].

A HiP-HOPS study of a system design typically has three main phases:

- Modelling Phase: system modelling & failure annotation
- Synthesis Phase: fault tree synthesis
- Analysis Phase: fault tree analysis & FMEA synthesis

Although the first phase remains primarily manual in nature, the other phases are fully automated. The general process in HiP-HOPS is illustrated in Figure 2 below:

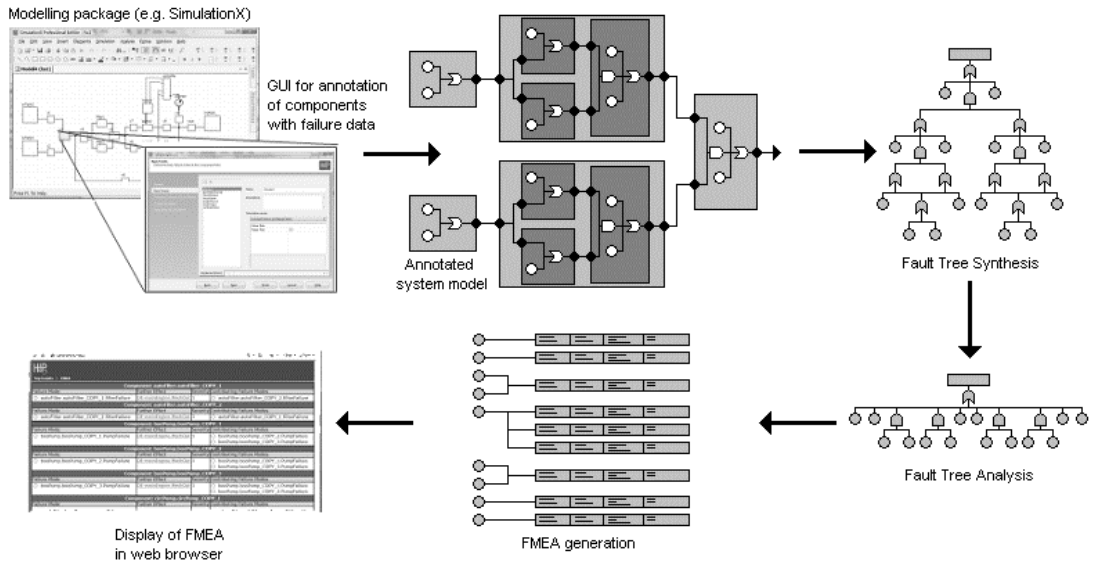


Figure 2: Overview of the HiP-HOPS Process

The first phase – system modelling & failure annotation – consists of developing a model of the system (including hydraulic, electrical or electronic, mechanical systems, as well as conceptual block and data flow diagrams) and then annotating the components in that model with failure data. This phase is carried out using an external modelling tool or package compatible with HiP-HOPS. HiP-HOPS has interfaces to a number of different modelling tools, including Matlab Simulink, Eclipse-based UML tools, and particularly SimulationX. The latter is an engineering modelling & simulation tool developed by ITI GmbH [37] with a fully integrated interface to HiP-HOPS. This has the advantage that existing system models, or at least models that would have been developed anyway in the course of the design process, can also be reused for safety analysis purposes rather than having to develop a new model specific to safety.

The second phase is the fault tree synthesis process. In this phase, HiP-HOPS automatically traces the paths of failure propagation through the model by combining the local failure data for individual components and subsystems. The result is a network of interconnected fault trees defining the relationships between failures of system outputs and their root causes in the failure modes of individual components. It is a deductive process, working backwards from the system outputs to determine which components caused those failures and in what logical combinations.

The final phase involves the analysis of those fault trees and the generation of an FMEA. The fault trees are first minimised to obtain the *minimal cut sets* – the smallest possible combinations of failures capable of causing any given system failure – and these are then used as the basis of both quantitative analysis (to determine the probability of a system failure) and the FMEA, which directly relates individual component failures to their effects on the rest of the system. The FMEA takes the form of a table indicating which system failures are caused by each component failure.

The various phases of a HiP-HOPS safety analysis will now be described in more detail.

4.1 Modelling Phase

HiP-HOP studies can be performed on any model of a system that identifies components and the material, energy or data transactions among those components. Typically these models will have been produced in a modelling tool such as Matlab Simulink or SimulationX. In addition, the models can be defined hierarchically, using a composition of components and subsystems, to help manage complexity. Failures can therefore propagate 'vertically' through the hierarchy as well as horizontally through the system.

Once a model has been obtained, it is necessary to annotate it with failure data. At its core, HiP-HOPS operates on the idea that an output failure of a component is caused by a logical combination of input failures and internal faults and that the output failure will then propagate along structural connections in the model to another component to be received as a new input failure. Thus for the purposes of the safety analysis, each component in the model needs to be annotated with its own local failure data, describing how that component can fail and how it responds to failures propagated from other components in the system.

The local failure data takes the form of a set of failure expressions relating failures at a component's outputs (known as *output deviations*) to a logical combination of internal failure modes (*basic events*) and *input deviations* (i.e. failures received at the component's inputs). For the specification of input and output deviations, a generic and abstract syntax was developed, consisting of two parts; the first part is the *failure class*, an identifier that describes the type of failure, and the second is the input or output *port* at which the failure is received or propagated. There are different ways of classifying failures, e.g. by relating them to the function of the component, or by classifying according to the degree of failure – complete, partial, intermittent etc [38]. In general, however, the failure of a component will have adverse local effects on the outputs of the component which, in turn, may cause further effects travelling though the system on material, energy or data exchanged with other components. Therefore in HiP-HOPS, we generally classify the effects into one of three main failure classes, all equally applicable to material, energy or data outputs: *omissions*, i.e. the failure to provide the input or output; *commissions*, i.e. a condition in which the input or output is provided inadvertently and in the wrong context of operation; and finally *malfunctions*, a general condition in which the input or output is provided but in a form which deviates from the design intention, e.g. with a value that exceeds

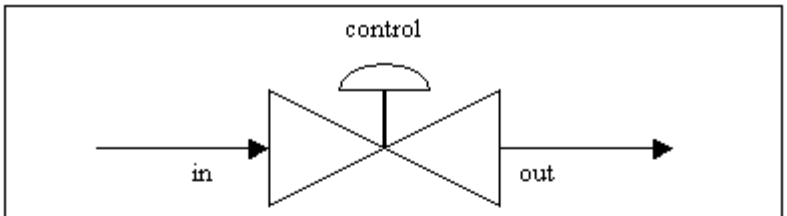
thresholds or is transmitted at the wrong time. Since this classification adopts a functional viewpoint which is independent of technology, it could provide a common basis for describing component failures and their local effects. However, HiP-HOPS can work with any classification of failures as long as it is used consistently from one component to the next, and indeed it allows users to define their own failure classes.

In HiP-HOPS, failure classes are often abbreviated, e.g. O = Omission, C = Commission, V = Value etc, and combined with the name of the port at which they occur, thus "O-input1" might be an omission of input at port "input1". Sometimes it is useful to parameterise the failure class as well, e.g. OF for omission of flow or HP for high pressure etc. HiP-HOPS also makes use of the standard Boolean logic operators AND and OR to combine input deviations and basic events and then relate these to a given output deviation, e.g. "O-out1 = O-in1 OR internalFailure" is an expression that describes how an omission of output is caused by a corresponding omission of input or some internal failure of the component itself. Internal failure modes typically depend on the domain, e.g. a blockage for a hydraulic system or a short circuit in an electrical system etc. Note that failure classes can also be transformed from input to output; for example, if a particular component was designed to fail silent in response to input errors, it may transform value input failures into an omission of output: "O-out = V-in OR C-in" etc. In this way, mitigation of failures can also be represented.

The set of failure expressions for a component therefore describes all possible deviations of all outputs for that component in terms of its possible internal failure modes and any relevant deviations at its inputs (which are in turn propagated from output deviations of other components). In addition to the logical information, it is possible to add numerical data for the failure modes of the component, detailing the probability of the occurrence each failure. HiP-HOPS allows multiple different probabilistic models to be used, e.g. constant failure & repair rates, MTTF & MTTR values, Binomial and Poisson distributions, dormant failures, and Weibull variable failure rates etc. This provides a great deal of flexibility when modelling the quantitative aspects of component failure (assuming the probability data for the failure modes are available). This data is later used to arrive at an estimate of the unavailability for each system failure.

The local failure data – both the logical expressions and the probabilistic data – can also be stored in a component library, allowing it to be reused in other system models and thereby reducing the time needed for failure annotations of future systems. Since the failure data is local to each component, there are no dependencies on other components that would complicate their reuse.

As an example of local component failure data in HiP-HOPS, Figure 3 below shows an analysis of a computer controlled valve. The figure shows the valve as it might be illustrated in a plant diagram and records the results of the local safety analysis of the component in two tables that define the internal failure modes of the valve and its output deviations respectively.



INTERNAL FAILURE MODES

Failure Mode	Description	Failure Rate	Repair Rate
blocked	e.g. by debris	1e-6	1e-4
partiallyBlocked	e.g. by debris	5e-5	1e-4
stuckClosed	Mechanically stuck	1.5e-6	1e-4
stuckOpen	Mechanically stuck	1.5e-5	1e-4

OUTPUT DEVIATIONS

Output Deviation	Description	Logical Causes
Omission-out	Omission of flow from output	blocked OR stuckClosed OR Omission-in OR Low-control
Commission-out	Unexpected flow from output	stuckOpen OR Commission-in OR High-control
LowFlow-out	Low flow rate at output	partiallyBlocked OR Low-in

Figure 3: Local failure data for a valve

In normal operation, the valve is normally closed and opens only when the computer control signal has a continuously maintained value of a logical one. Valve malfunctions include mechanical failures such as the valve being stuckOpen or stuckClosed, and blockages caused by debris such as blocked and partiallyBlocked. For each malfunction, the analysis records estimated failure and repair rates while the effects of those malfunctions on the output of the valve can be seen in a second table that lists output deviations.

This specification of failure modes is generic in the sense that it does not contain references to the context within which the valve operates. Failure expressions make references only to internal malfunctions and input/output ports of the component. The failure behaviour described in these expressions has been derived assuming a simple function that we expect the component to perform in every application (valve is normally closed unless the value of control signal is 1). For these reasons, the specification of the valve in the figure above provides a template that could be re-used in different models and contexts of operation, perhaps with some modifications, e.g. on failure rates, to reflect a different environment.

4.2 Synthesis Phase

As explained above, the local failure data for each component relates deviations of its outputs to combinations of input deviations and internal component malfunctions – effectively a set of small fault trees describing the failures of the component. When we examine a component out of system context, input and output deviations represent only potential conditions of failure. However, when we place the component in a model of a system, the input deviations specified in the analysis can actually be triggered by other components further upstream in the model and the specified output deviations can similarly cause more failures further downstream.

Thus by linking the output failures of a certain class from one component to the input failures of that same failure class at another component, via the architectural connections and interactions stored in the model, it is possible to map out the global propagation of failures through the system as a whole in the form of a series of interconnected fault trees. Thus the causes of a significant hazard or failure at the

output of the system can be traced back through this propagation by simply analysing the fault trees and determining the minimal cut sets.

This process of synthesising fault trees from the component failure data (which are effectively mini fault trees) is automated by HiP-HOPS. The fault trees are constructed incrementally, working backwards from the outputs of the system (e.g. electromechanical actuators) towards the system inputs (e.g. material/energy resources, operators, and data sensors etc) and joining causes of failures in one component to their effects in another. At each stage, the mini fault trees representing the local failure data of a component are added to the tree being generated; note that HiP-HOPS will allow branches of the fault tree to be shared if they are traversed more than once, rather than duplicating them. Figure 4 below illustrates this process, showing how the local failure data is connected together to form a larger system fault tree.

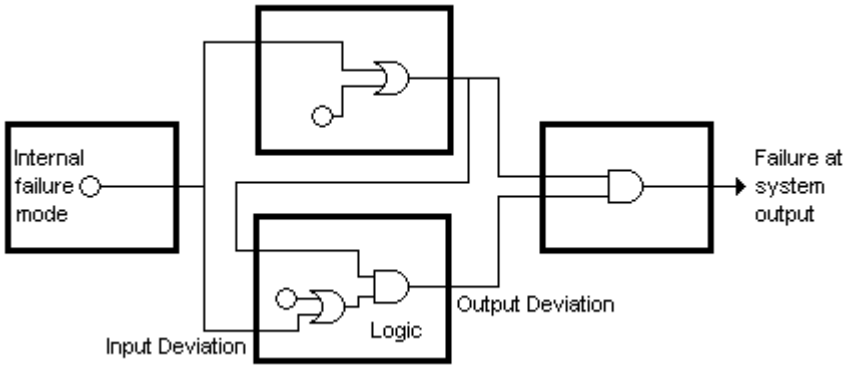


Figure 4: Connecting local failure data together to synthesise fault trees

In this way, an overall view of the global propagation of failure in the system can be automatically captured by traversing the model and by following the causal links specified in the local safety analyses of the components. Note that the mechanically synthesised fault trees produced record the propagation of failure in a very strict and methodical way, starting from an output failure and following dependencies between components in the model to systematically record other component failures that progressively contribute to this event. The logical structure of the tree is determined only by interconnections between the components and the local analyses of those components. This logical structure is straightforward and can be easily understood, unlike the structure of many manually constructed fault trees, which is often defined by implicit assumptions made by analysts.

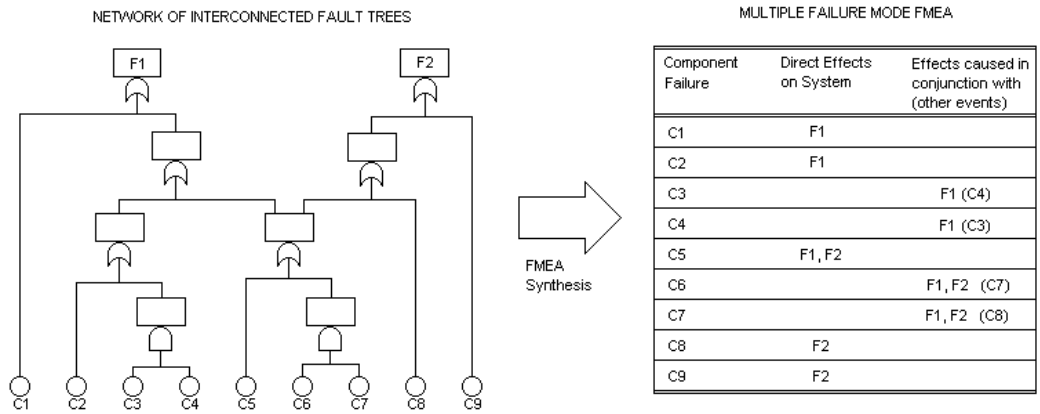
To manage complex hierarchical models effectively, the synthesis algorithm in HiP-HOPS will perform traversals both across the vertical and horizontal axis of the design hierarchy, allowing the annotation of the system hierarchy at all levels of the design. If, for example, a subsystem as a whole is susceptible to a failure mode, then the effect of this condition can be directly specified with a failure annotation at subsystem level. This annotation, for example, could define that all outputs of the subsystem are omitted in the event of a global disturbance, e.g. in the case of electromagnetic interference. Such annotations would typically complement other annotations made at the level of the enclosed components to describe aspects of failure behaviour at this level (e.g. the mechanical and electrical failure modes of each component). In general, when examining the causes of a failure at an output of a subsystem, the fault tree synthesis algorithm creates a disjunction between any failure logic specified at subsystem level and logic arising from the enclosed lower levels.

HiP-HOPS is also designed to recognise and handle loops in the model that create circular references to the same failure logic (e.g. conditions such as: event A is caused by event B which in turn is caused by event A). When such circles are encountered, the failure logic contained in the circle is only incorporated once in the fault trees. At the same time, a warning note is generated in order to encourage the analyst to investigate whether or not the circular logic in the system was valid or the result of a modelling error. In general it is normally possible in the steady state to determine what the final effect on the system will be of an initiating failure further upstream.

4.3 Analysis Phase

In the final phase, the synthesised system fault trees are analysed, both qualitatively and quantitatively, and from these results a multiple failure mode FMEA is generated. Firstly, the fault trees undergo qualitative analysis to obtain their minimal cut sets, which reduces them in size and complexity. This is achieved using various logical reduction techniques, i.e. applying logical rules to reduce the complexity of the expressions and remove any redundancies. Once the minimal cut sets have been obtained, they are analysed quantitatively, which produces unavailability values for the top events of each fault tree.

The last step is to combine all of the data produced into an FMEA, which is a table that concisely illustrates the results. The FMEA shows the direct relationships between component failures and system failures, and so it is possible to see both how a failure for a given component affects everything else in the system and also how likely that failure is. A classical FMEA only shows the *direct effects* of single failure modes on the system, but because of the way this FMEA is generated from a series of fault trees, HiP-HOPS is not restricted in the same way, and the FMEAs produced also show what the *further effects* of a failure mode are; these are the effects that the failure has on the system when it occurs in conjunction with other failure modes. Figure 5 below shows this concept:



Component Failure	Direct Effects on System	Effects caused in conjunction with (other events)
C1	F1	
C2	F1	
C3		F1 (C4)
C4		F1 (C3)
C5	F1, F2	
C6		F1, F2 (C7)
C7		F1, F2 (C8)
C8	F2	
C9	F2	

Figure 5: Converting networks fault trees into a multiple failure mode FMEA

In the figure, F1 and F2 are system failures, and C1 – C9 are component failures. For C3, C4, C6 and C7, there are no direct effects on the system – that is, if only one of these components fail, nothing happens. However, they do have further effects; for example, C3 and C4 both occurring in conjunction will cause F1 to occur. The FMEAs produced thus show all of the effects on the system, either singly or in combination, of a particular component failure mode. This is especially useful because it allows the designer to identify failure modes that contribute to multiple system failures (e.g. C5 in the example). These common cause failures represent especially vulnerable points in the system, and are prime candidates for redundancy or substitution with more reliable components.

4.4 Tool support

Although HiP-HOPS has interfaces with a number of different modelling tools, the most advanced, fully-featured interface is with the SimulationX modelling & simulation tool from ITI GmbH. A commercial version of HiP-HOPS is also available with SimulationX.

Simulation X provides a Graphical User Interface (GUI) that enables annotation of components in the model with the failure modes and failure expressions required for the fault tree and FMEA synthesis. The data becomes part of the model and is automatically saved and retrieved by SimulationX. Failure annotations are stored together with the component in component libraries and can be re-used either directly or following modifications within the same model or across different models with the obvious benefit of simplifying the manual part of the analysis. When an analysis takes place, SimulationX generates an output file that is parsed by HiP-HOPS, which then reconstructs the enclosed annotated models for the purposes of fault tree synthesis and analysis. After HiP-HOPS analyses the fault trees, it generates output files containing the fault trees, the cut sets and unavailability (if available), and the direct and further effects FMEA tables. The direct effects table is a single failure mode FMEA which shows, for each failure mode of each component in the system, the direct effects on the system. Further effects caused by conjunctions of component failure modes are shown in a second table which presents a lengthier and more detailed multiple failure mode FMEA analysis. The output is created by a HTML generator, which produces web pages containing the tables of data. The advantages of this medium

include easy distribution and display and the ability, through hyperlinks, to navigate different aspects of the information. The various interfaces of the tool are shown below in Figure 6:

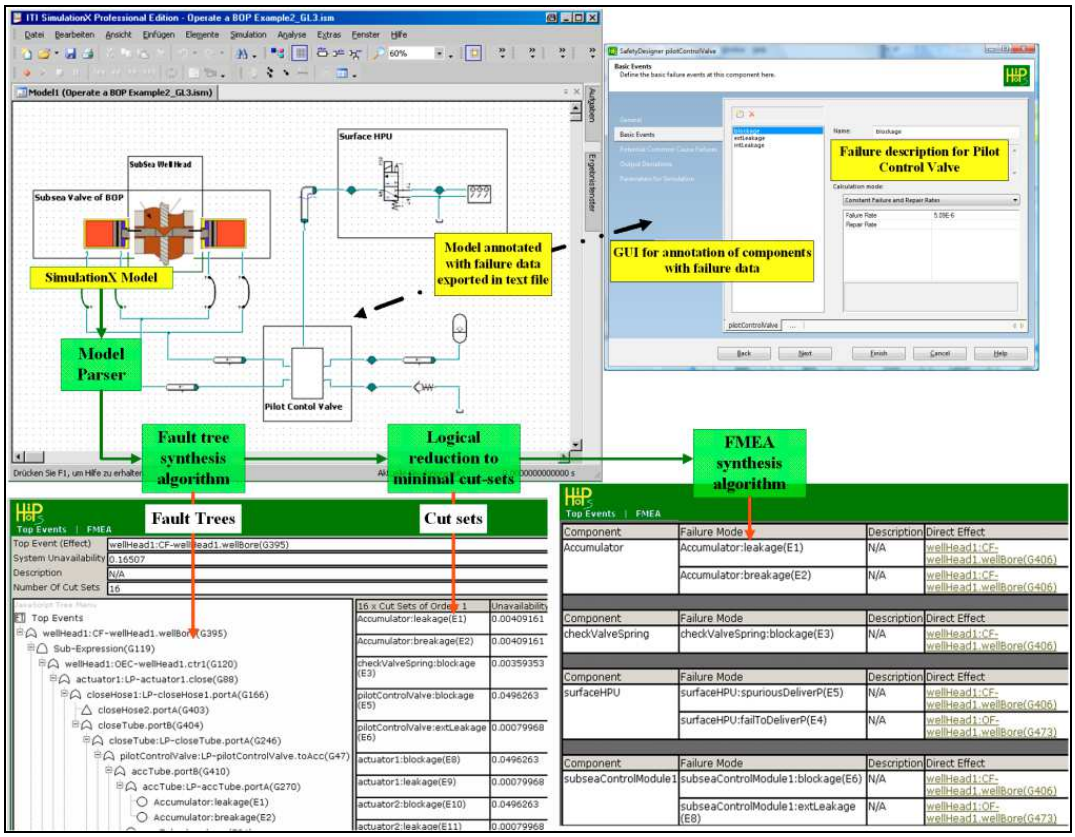


Figure 6: SimulationX & HiP-HOPS tool interfaces

5 Design Optimisation using HiP-HOPS

HiP-HOPS analysis may show that safety, reliability and cost requirements have been met, in which case the proposed system design can be realised. In practice, though, this analysis will often indicate that certain requirements cannot be met by the current design, in which case the design will need to be revised. This is a problem commonly encountered in the design of reliable or safety critical systems. Designers of such systems usually have to achieve certain levels of safety and reliability while working within cost constraints. Design is a creative exercise that relies on the technical skills of the design team and also on experience and lessons learnt from successful earlier projects, and thus the bulk of design work is creative. However, we believe that further automation can assist the process of iterating the design by aiding in the selection of alternative components or subsystem architectures as well as in the replication of components in the model, all of which may be required to ensure that the system ultimately meets its safety and reliability requirements with minimal cost.

A higher degree of reliability and safety can often be achieved by using a more reliable and expensive component, an alternative subsystem design (e.g. a primary/standby architecture), or by using replicated components or subsystems to achieve redundancy and therefore ensure that functions are still provided when components or subsystems fail. In a typical system design, however, there are many options for substitution and replication at different places in the system and different levels of the design hierarchy. It may be possible, for example, to achieve the same reliability by substituting two sensors in one place and three actuators in another, or by replicating a single controller or control subsystem etc. Different solutions will, however, lead to different costs, and the goal is not only to meet the safety goals and cost constraints but also to do so optimally, i.e. find designs with maximum possible reliability for the minimum possible cost. Because the options for replication and/or substitution in a non-trivial design are typically too many to consider manually, it is virtually impossible for designers to address this problem systematically; as a result, they must rely on intuition, or on evaluation of a few different design options. This means that many other options – some of which

are potentially superior – are neglected. Automation of this process could therefore be highly useful in evaluating a lot more potential design alternatives much faster than a designer could do so manually.

Recent extensions to HiP-HOPS have made this possible by allowing design optimisation to take place automatically [39]. HiP-HOPS is now capable of employing genetic algorithms in order to progressively "evolve" an initial design model that does not meet requirements into a design where components and subsystem architectures have been selected and where redundancy has been allocated in a way that minimizes cost while achieving given safety and reliability requirements. In the course of the evolutionary process, the genetic algorithm typically generates populations of candidate designs which employ user-defined alternative implementations for components and subsystems as well as standard replication strategies. These strategies are based on widely used fault tolerant schemes such as hot or cold standbys and n-modular redundancy with majority voting.

For the algorithm to progress towards an optimal solution, a selection process is applied in which the fittest designs survive and their genetic makeup is passed to the next generation of candidate designs. The fitness of each design relies on cost and reliability. To calculate fitness, therefore, we need ways in which to automatically calculate those two elements. An indication of the cost of a system can be calculated as the sum of the costs of its components (although for more accurate calculations, life-cycle costs should also be taken into account, e.g. production, assembly and maintenance costs) [40]. However, while calculation of cost is relatively easy to automate, the automation of the evaluation of safety or reliability is more difficult as conventional methods rely on manual construction of the reliability model (e.g. the fault tree, reliability block diagram or the FMEA). HiP-HOPS, by contrast, already automates the development and calculation of the reliability model, and therefore facilitates the evaluation of fitness as a function of reliability (or safety). This in turn enables a selection process through which the genetic algorithm can progress towards an optimal solution which can achieve the required safety and reliability at minimal cost.

One issue with genetic algorithms is that it has to be possible to represent the individuals in the population – in this case, the design candidates – as genetic encodings in order to facilitate crossover and mutation. Typically this is done by assigning integers to different alternatives in specific positions in the encoding string, e.g. a system consisting of three components may be represented by an encoding string of three digits, the value of each of which represents one possible implementation for those components. However, although this is sufficient if the model has a fixed, flat topology, it is rather inflexible and cannot easily handle systems with subsystems, replaceable sub-architectures, and replication of components, since this would also require changing the number of digits in the encoding string.

The solution used in HiP-HOPS is to employ a *tree encoding*, which is a hierarchical rather than linear encoding that can more accurately represent the hierarchical structure of the system model. Each element of the encoding string is not simply just a number with a fixed set of different values, it can also represent another tree encoding itself. Figure 7 shows these different possibilities: we may wish to allow component A to be replaced with either a low cost, low reliability implementation (represented as 1), a high cost, high reliability implementation (2), or an entirely new subsystem with a primary/standby configuration (3). If the third implementation is selected, then a new sub-encoding is used, which may contain further values for the components that make up the new subsystem, i.e. the primary and the standby:

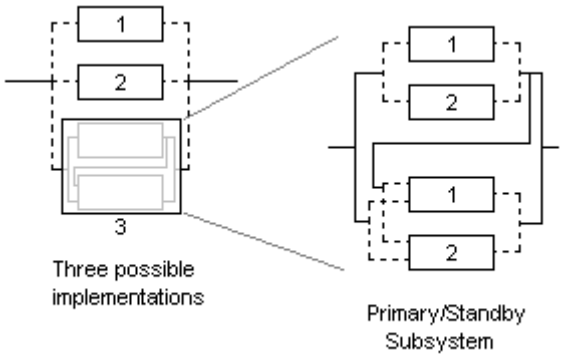


Figure 7: Three different implementations of a component

Thus encoding "1" means that the first implementation was chosen, encoding "2" means the second was chosen, "3(11)" means that the third was chosen (the subsystem) and furthermore that the two subcomponents both use implementation 1, while "3(21)" for example means that the primary component in the subsystem uses implementation 2 instead. Although the tree encoding is more complex, it is also much more flexible and allows a far greater range of configuration options to be used during the optimisation process.

HiP-HOPS uses a variant of the NSGA-II algorithm for optimisation. The original NSGA-II algorithm allows for both undominated and dominated solutions to exist in the population (i.e. the current set of design candidates). To help decide which solutions pass on their characteristics to the next generation, they are ranked according to the number of other solutions they dominate. The more dominant solutions are more likely to be used than the less dominant solutions. HiP-HOPS is also able to discard all but the dominant solutions. This is known as a pure-elitist algorithm (since all but the best solutions are discarded) and also helps to improve performance.

To further enhance the quality of solutions and the speed with which they can be found, a number of other modifications were made. One improvement was to maintain a solution archive similar to those maintained by tabu search and ant colony optimisation; this has the benefit of ensuring that good solutions are not accidentally lost during subsequent generations. Another improvement was to allow constraints to be taken into account during the optimisation process, similar to the way the penalty-based optimisation functions: the algorithm is encouraged to maintain solutions within the constraints and solutions outside, while permitted, are penalised to a varying degree. In addition, younger solutions – i.e. ones more recently created – are preferred over ones that have been maintained in the population for a longer period; again, this helps to ensure a broader search of the design space by encouraging new solutions to be created rather than reusing existing ones.

6 Example: Optimisation of a marine fuel oil system

To demonstrate the application of design optimisation in practice, we applied the optimisation capabilities of HiP-HOPS to a simplified fuel oil service system for a cargo ship. When the fuel oil system fails, there is a loss of engine propulsion that can lead to the ship becoming grounded as a result of drifting. The base design – with no component replication or substitution – is shown in Figure 8 below:

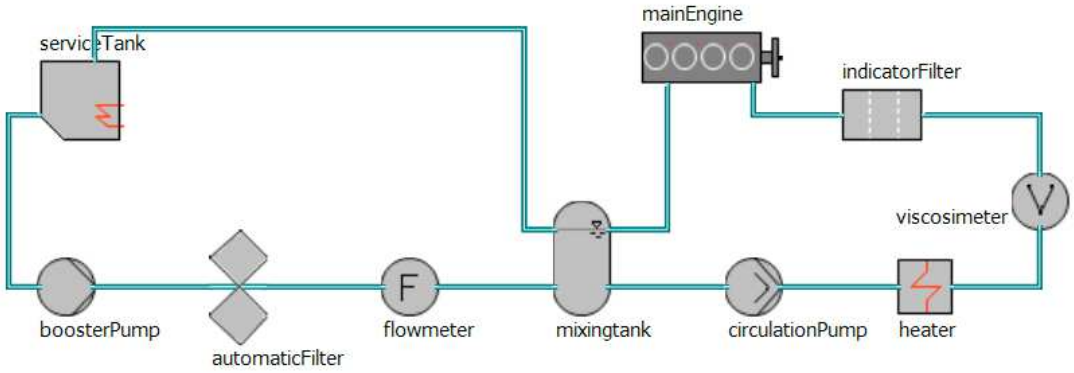


Figure 8: Fuel Oil system – basic design

This system was annotated with failure data and cost information in the SimulationX interface, allowing a standard HiP-HOPS analysis to take place. It was also further augmented with information to define the design search space, by specifying which components could be replicated (all but the main engine). For our preliminary experiments, we restricted the potential optimisations types to simple replication to achieve a maximum of triple redundancy (i.e. 0, 1, or 2 replicants); as a result, only a single set of cost & failure data was needed (since each replication is a simple duplication). This data is displayed in the table below.

Table 1: Cost & Failure data for the Fuel Oil system

Components	Cost	Failure Rate

Indicator filter	1500	5.0E-7
Viscosimeter	2500	2.5E-6
Pre-heater	2000	6.7E-6
Circulation pump	6000	3.2E-5
Mixing tank	2000	1.6E-5
Flow meter	2000	1.0E-5
Automatic filter	2000	1.0E-5
Booster pump	5000	3.2E-5
Service tank	1500	1.6E-5

At first a manual attempt at optimisation was made by the designer. Even with the scope for optimisation being restricted to only replication of components, the search space still contains 19683 possible solutions; only 12 were actually considered during the manual analysis. HiP-HOPS then performed a normal reliability and cost analysis on each of these solutions, and then the designer carried out a cost-benefit analysis by weighing the total system cost of each solution against the cost of a grounding occurring due to failure of the fuel oil system (this is the product of the system unavailability and the physical cost of losing the ship). The system with the best cost-benefit ratio was selected as the reference solution, shown below in Figure 9:

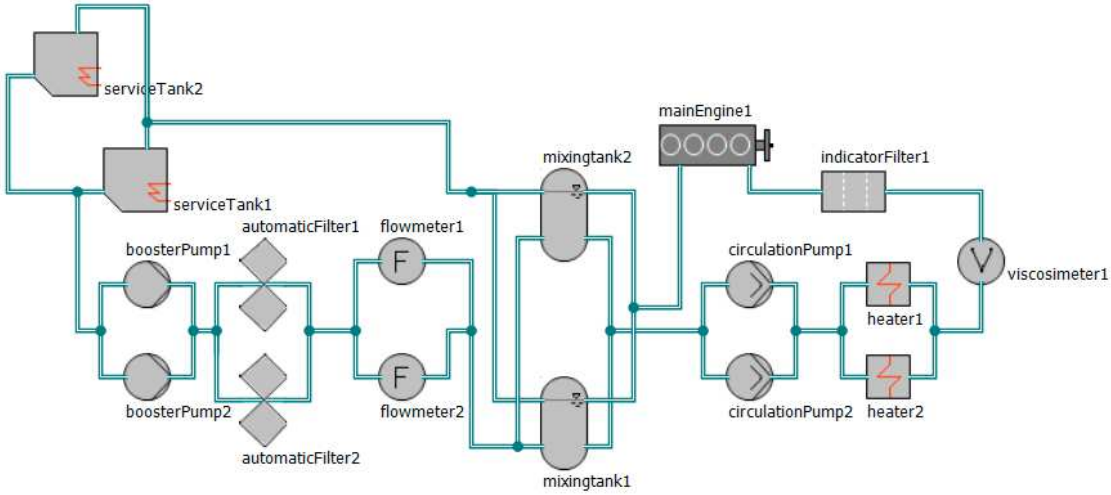


Figure 9: Fuel Oil system – manually optimised design

This system has a cost of 45000 and an unavailability of $3x10^{-6}$. This then established a baseline to compare with the results of the automatic optimisation.

Unlike the manual optimisation, HiP-HOPS was not restricted to investigating a small number of candidates. 50 generations of optimisation took place, discovering 46 non-dominated (Pareto) solutions. These 46 solutions are plotted in a graph in Figure 10 below (as crosses), alongside the 12 manually investigated solutions (circles).

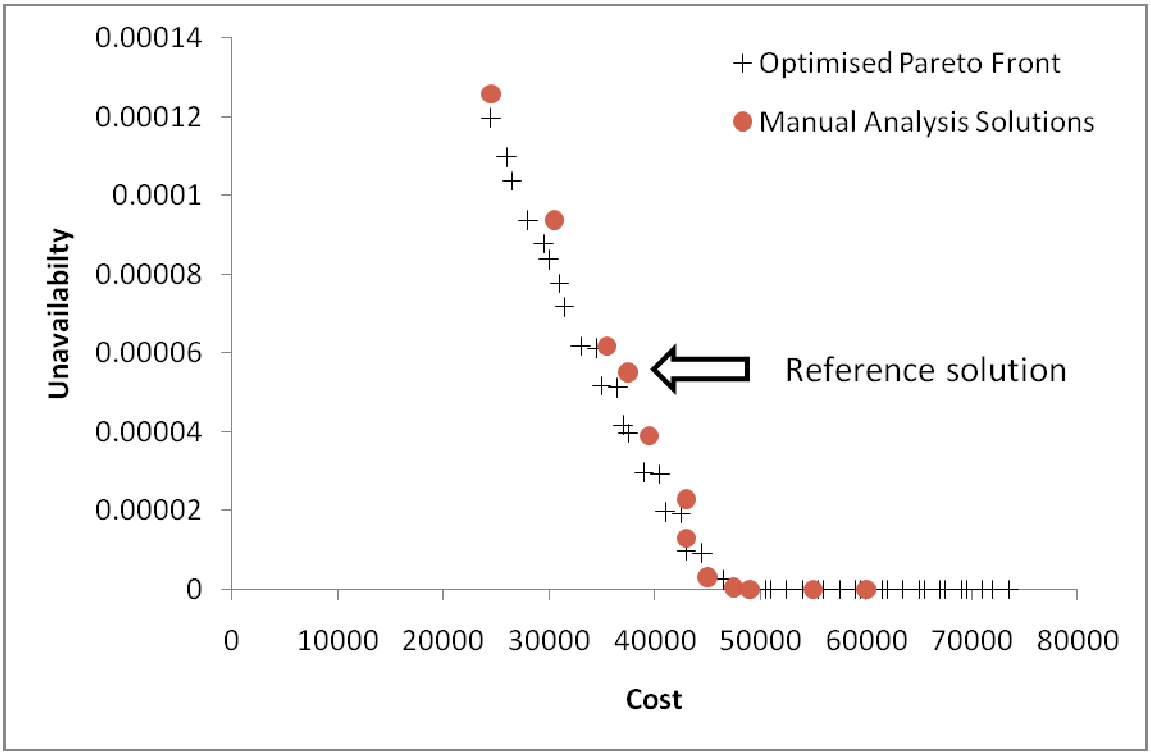


Figure 10: Plotting the Pareto Front

As can be seen, most of the manually investigated solutions are dominated by the Pareto solutions discovered by the automatic optimisation process. Another cost-benefit analysis was then carried out on these solutions to determine which offered best value for money.

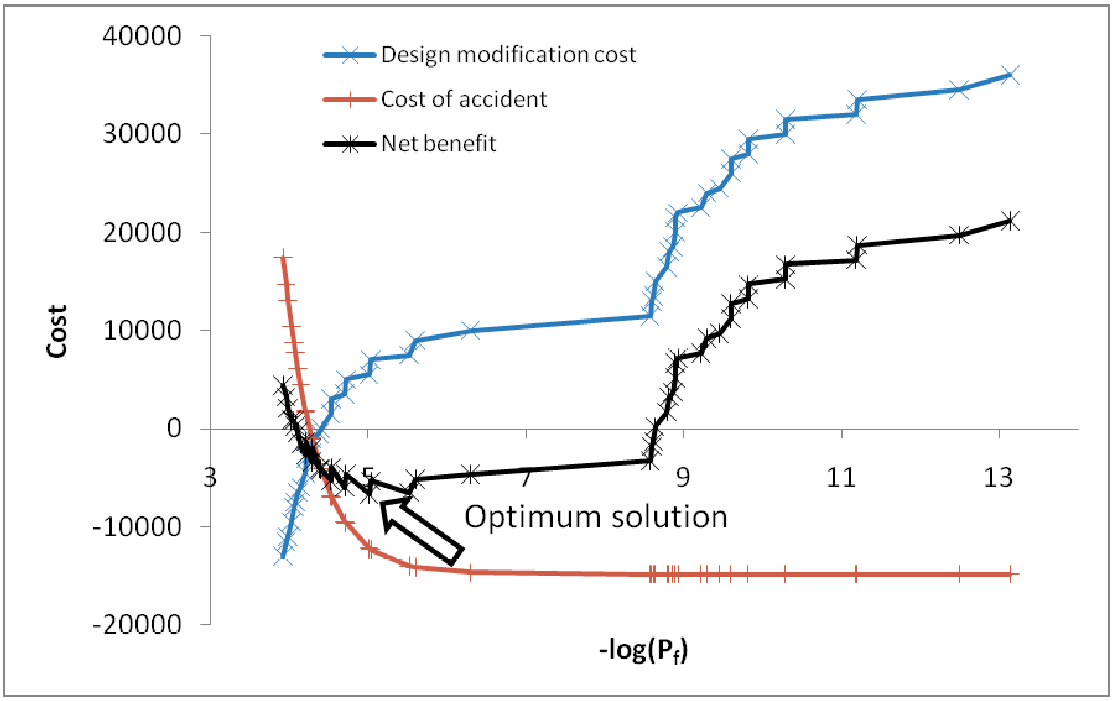


Figure 11: Cost-benefit analysis of the automatic optimisation solutions

The optimal solution (offering the lowest cost) is highlighted. It has a slightly higher unavailability (9.7×10^{-6}) but the cost is only 43000 as opposed to 45000. The difference is that only one heater is used, rather than the two found in the reference solution shown in Figure 11.

Next we lifted the restrictions on the optimisation process, allowing 3 different alternative implementations of each of the components to be selected, each with different costs and failure rates. This increased the search space to 20,661,046,784 different possible solutions. Clearly, the percentage of these solutions that could be investigated manually is extremely small.

Automatic optimisation using HiP-HOPS discovered 366 non-dominated solutions in 4000 generations, plotted in Figure 12 below:

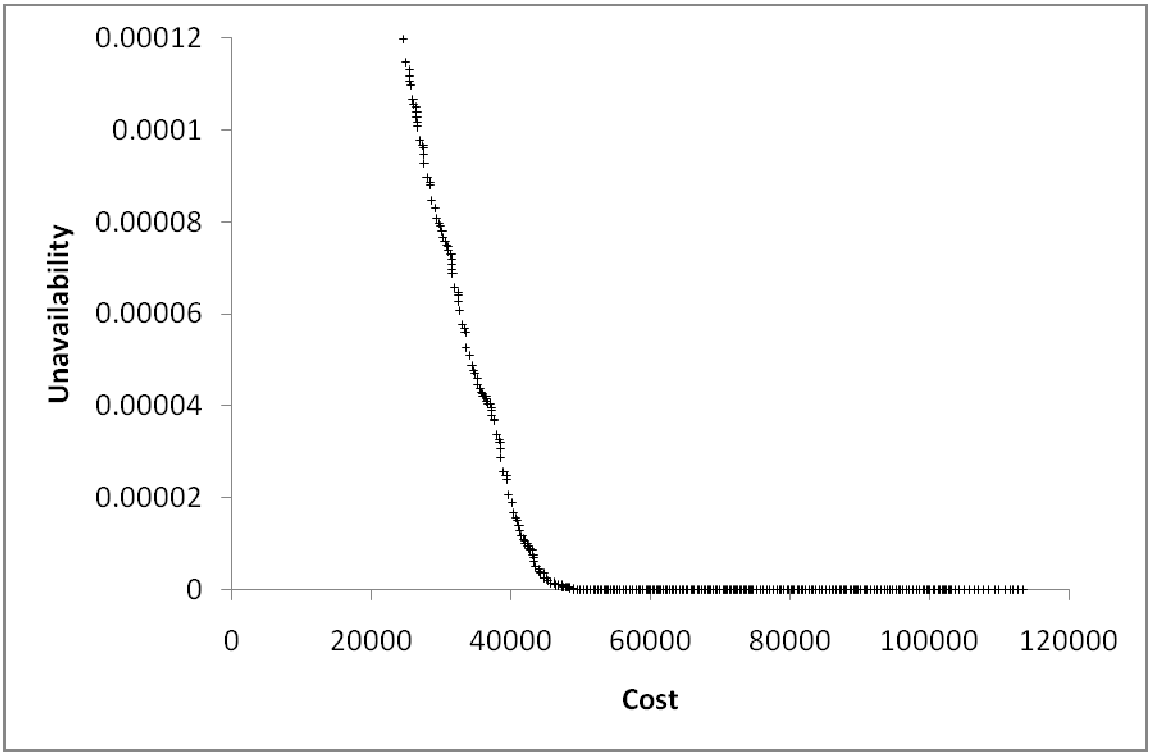


Figure 12: Plotting the larger Pareto Front

As before, a cost benefit analysis was carried out on the Pareto solutions. The results of that analysis are shown in Figure 13 below; the chart has been zoomed in to show detail around the optimum area.

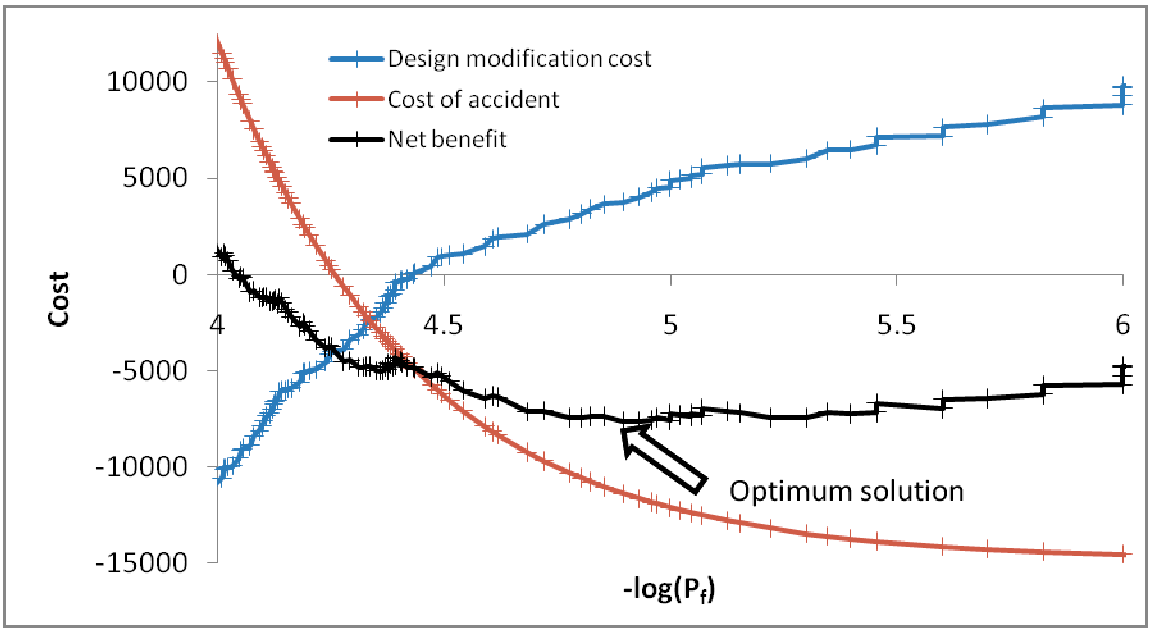


Figure 13: New cost-benefit analysis

Being able to consider alternative components for each of the subsystems allows a new optimum configuration. This is shown in Figure 14 below, where the automatic filter and the flow meter have a single component, alternative version 3 and 2 respectively, in place of a redundant configuration.

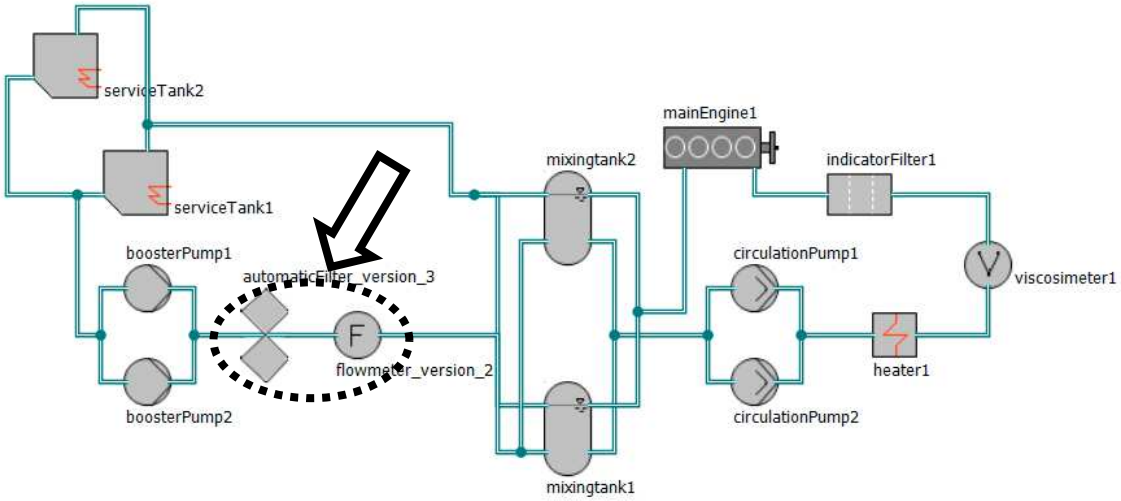


Figure 14: Fuel Oil system – optimal design

The system's unavailability of 1.17×10^{-5} is worse than the previous two best solutions, but the component cost is only 41529.

Although the above is only a relatively small and simple example, the search space for even this system is still prohibitively large; it would not be possible to explore more than a tiny fraction of it using a manual optimisation process, and thus it is unlikely that the designer would have been able to identify any truly optimal solutions. By contrast, automatic optimisation enabled the designer to explore many more solutions and resulted in a set of optimal trade-offs between reliability and cost, as shown above. It is important to note that while only 12 solutions were considered manually, the automatic optimisation returned 366 optimal solutions, which in turn is only a small fraction of the total number of solutions evaluated. By performing a cost-benefit analysis on these results, it is possible for the designer to determine which is likely to be the most suitable design; in both the restricted and non-restricted optimisation, HiP-HOPS discovered a better solution than any found by the manual analysis. The solution shown above offers a 17% improvement in net benefit over the manual reference solution.

7 Comparison with state of the art

7.1 Safety Analysis

As described in section 2, HiP-HOPS is a compositional safety analysis technique. It is closely analogous to the CFT and FPTC approaches, although unlike FPTC, HiP-HOPS is a deductive technique and is thus less prone to combinatorial explosion. It currently lacks the ability to analyse systems with states and state transitions as in SEFTs, though there have been recent developments in adding temporal logic capabilities to HiP-HOPS, but it is capable of dealing with NOT gates and non-coherent fault trees [41]. However, unlike SEFTs, HiP-HOPS can make use of well-tested, high performance FTA algorithms since it does not need to take into account the complexities of dynamic state transitions in the system. Like most of these other approaches (but not FPTN), HiP-HOPS also benefits from making use of the architectural design model rather than having to build a separate error model for the purposes of safety analysis. One of the major advantages of HiP-HOPS, however, is the type of output it produces; HiP-HOPS does not rely on external tools or other representations like Petri Nets to perform the analysis, and instead produces both fault trees and FMEAs directly. The fault tree analysis results include both the fault trees themselves as well as their cut sets and unavailability estimates for the system failures, while the FMEA results include not only the direct effects of failures, but also the further effects they may have in conjunction with other failures.

Compared to the formal model checking & fault simulation approaches like Altarica and FSAP-NuSMV etc, HiP-HOPS requires a manual annotation phase before analysis & synthesis can take place; however, the benefit of this is that more sophisticated failure logic can be modelled and the system designer can ensure that the failure behaviour in the model is a more accurate reflection of the design intentions. In addition, because the fault simulation model is the same as the nominal system model, both are typically specified in a custom modelling language which is not always compatible with other design languages and tools. Since HiP-HOPS focuses only on failure modelling, it can more easily complement other design languages and tools that focus primarily on nominal behaviour. It is because of this that HiP-HOPS can easily be integrated with simulation tools like SimulationX.

The other advantage of HiP-HOPS over the simulation approaches is related to its optimisation capabilities. During optimisation, tens and even hundreds of thousands of different solutions need to be evaluated quickly, meaning that tool performance is critical. Fault simulation techniques are primarily inductive in nature, looking at the effects on system behaviour caused by the injection of single faults; this is typically slower than deductive methods like HiP-HOPS that work backwards from system failures to determine their causes, particularly for larger systems where combinatorial explosion is more of an issue. Although fault tree analysis can still be time consuming, HiP-HOPS can usually analyse even complex models with thousands of failure modes in a few seconds, which is critical for use in optimisation where execution times are magnified a thousand-fold due to the number of iterations involved.

Nevertheless, this does not mean that other techniques have no place. For example, HiP-HOPS could first be used to identify potential design candidates using deductive safety analysis techniques and automatic optimisation, and then a selection of the optimal solutions could be subjected to a more rigorous simulation to better understand their nominal and failure behaviour.

7.2 Optimisation

In addition to its safety analysis features, HiP-HOPS has recently been extended with automatic architectural optimisation capabilities, allowing system designs to be evolved with respect to reliability and cost. It has so far been tested on both established benchmark problems [42] and smaller case studies like the fuel oil system presented above. Although it is too premature to comment on overall performance, as its scalability to larger, real-world systems is still being investigated, the successful results so far indicate that combining automatic evolutionary algorithms for design optimisation with modern safety analysis techniques may offer significant benefits for safety-driven design of critical systems.

By contrast to the integrated safety analysis & optimisation capabilities found in HiP-HOPS, other approaches to architectural optimisation [42-44] typically calculate reliability from manually constructed Reliability Block Diagrams (RBDs), in which systems are represented as series-parallel configurations of components that possess a single failure mode and thus either function or fail. This is both an additional time-consuming modelling step and offers only limited expressive power to represent the real failure behaviour of the systems in question. In HiP-HOPS, the optimisation can

manipulate the actual system topology, within limits defined by the failure and optimisation parameters defined for each component, meaning that the architecture itself can change to make use of established design patterns like voters and redundant components in primary/standby configurations etc, in addition to the simple replication and substitution found in other approaches. This also means that the models are not limited to simple series-parallel configurations but can use more complex connection topologies and hierarchically nested subsystems. Furthermore, the safety analysis capabilities of HiP-HOPS are not limited to single failure modes and allow for a more sophisticated failure logic including many different classes of failure modes, including omissions, commissions, value errors and timing errors, that may be combined and transformed during their propagation through the system. These more realistic failure assumptions, we believe, will help to improve the quality of the solutions reached by this type of analysis.

8 Conclusion

Given the increasing scale and complexity of modern engineering systems, together with more stringent safety constraints and increasing pressures on design lifetimes and budgets, tool automation can offer significant benefits. Safety analysis tools can be a valuable aid in ensuring that system designs meet their safety goals, helping to identify potential flaws and weaknesses in early design iterations where corrective measures can more easily and more cheaply be carried out. The safety analysis features of HiP-HOPS make use of fast, deductive analysis techniques like FTA to automatically produce comprehensive information about the failure behaviour of a system. In addition, this analysis takes place on the actual architectural model of the system, although some additional annotation is required to detail the local failure data for the components in the system. This means that no separate error model is required and the failure data, once annotated, can be stored in normal component libraries and reused in different models and contexts with at most minor modifications, resulting in important savings in time and effort.

However, even though safety analysis tools like HiP-HOPS can help to ensure that safety is included as a driving factor in each iteration of the design, the iterations themselves remain a manual process, relying on the intuition and experience of the designers to evolve the system in a productive way. Design optimisation tools offer a way of accelerating this process by automatically investigating many more design possibilities than would be possible to evaluate manually; by adding some additional information to the system model to define the search space, modern evolutionary optimisation techniques like genetic algorithms can quickly and efficiently explore that search space to identify optimal solutions that feature improved trade-offs between conflicting objectives like cost and reliability. These solutions can then be investigated more closely by the designers and used as the basis for the next iteration of the system design.

The recent optimisation extensions to HiP-HOPS present a unique capability: by combining fast safety analysis abilities with state-of-the-art optimisation technology, it becomes possible to evaluate and optimise system models with regard to multiple objectives like cost and reliability within the same tool, resulting in a more integrated process with better performance. Although the technology is still experimental and only preliminary results have been obtained, they suggest that this type of combined analysis & optimisation automation potentially offers great advantages to designers of modern safety-critical systems. We hope to conduct further research to refine and enhance these optimisation capabilities to allow them to be scaled up to larger, more complex systems and to ensure they produce useful results for real-world engineering systems such as those found in the marine and automotive transport industries.

Acknowledgements

This work was supported by the EU Projects SAFEDOR (Grant 516278), ATESST2 (Grant 224442), and MAENAD (Grant 260057)

References

[1] Uhlig A., Kurzbach G., Hamann R., Papadopoulos Y., Walker M., Lühmann B., (2007) Simulation Model Based Risk and Reliability Analysis, Tagung Technische Zuverlässigkeit, Stuttgart, 22-23.April, Proceedings on CD, ISBN: 978-3-18-091984-3

[2] Hamann R., Uhlig A., Papadopoulos Y., Rüde E., Grätz U., Walker M., Lien R. (2008) Semi-automatic Failure Analysis Based on Simulation Models, Paper No. OMAE2008-57256, Proceeding OMAE 2008, Estoril, Portugal.

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

[4] U.S. Military (1949). Procedure for performing a failure mode effect and criticality analysis. United States Military Procedure MIL-P-1629.

[5] Isograph Software (2002). Fault Tree + v11. Software tool (http://www.isograph-software.com/index.htm)

[6] Worrell R.B., Stack D.W. (1978). A SETS User Manual for the Fault Tree Analyst. NUREG CR-04651, US Nuclear Regulatory Commission.

[7] Lisagor O., McDermid J.A., and Pumfrey D.J. (2006) Towards a practicable process for automated safety analysis, in Proceedings of the 16th International Ship and Offshore Structures Conference (ISSC'06).

[8] Fenelon P, McDermid J.A. (1993) An integrated toolset for software safety analysis. *Journal of Systems and Software*. **21**(3), pp.279-290.

[9] Wallace, M. (2005) Modular architectural representation and analysis of fault propagation. *Electronic Notes in Theoretical Computer Science*. **141**(3), pp.53-71.

[10] Ge X., Paige R.F., McDermid J.A. (2009) Probablistic Failure Propagation and Transformation Analysis. *Computer Safety, Reliability, and Security*. **5775**, pp.215-228.

[11] Grunske L., Kaiser B., Papadopoulos Y.I. (2005) Model-driven Safety Evaluation with State-event-based Component Failure Annotations. *In*: *8th International Symposium on Component-based Software Engineering*., pp.33-48.

[12] Kaiser B., Gramlich C., Forster M. (2007) State/event fault trees - A safety analysis model for software-controlled systems. *Reliability Engineering and System Safety*. **92**, pp.1521-1537.

[13] Grunske L., Kaiser B. (2005) An Automated Dependability Analysis Method for COTS-Based Systems. *Lecture Notes in Computer Science*., pp.178-190.

[14] Grunske L., Neumann R. (2002) Quality improvement by integrating non-functional properties in software architecture specification. *In*: *EASY'02 Second Workshop on Evaluating and Architecting System Dependability*. San Jose, California, USA, pp.23-32.

[15] Ciardo G., Lindermann C. (1993) Analysis of deterministic and stochastic Petri nets. *In*: *Proceedings of the 5th International Workshop on Petri nets and Performance models PNPM'93*. Toulouse, France.

[16] Zimmermann A., German R., Freiheit J., Hommel G. (1999) TimeNET 3.0 Tool Description. *In*: *International Conference on Petri Nets and performance models*. Zaragoza, Spain.

[17] Bozzano M., Villafiorita A. (2006) The FSAP/NuSMV-SA Safety Analysis Platform. *In*: *ECAI 2006*. Riva del Garda, Italy.

[18] Bieber P., Bougnol C., Castel C., Heckmann J-P., Kehren C., Metge S., Seguin C. (2004) Safety assessment with AltaRica. *In*: *IFIP Congress Topical Sessions*., pp.505-510.

[19] Griffault A., Arnold A., Point G., Rauzy A. (1999) The Altarica formalism for Describing Concurrent Systems. *Fundamenta Informaticae*. **34**.

[20] McMillan K.L. (1990) *Symbolic model checking*. Kluwer Academic Publishers.

[21] Heimdahl M.P., Choi Y., Whalen M.W. (2002) Deviation analysis though model checking. *In*: *Proceedings of the 17th IEEE International Conference on Automated Software Engineering*. Edinburgh, UK.

[22] Ortmeier F., Reif W., Schellhorn G. (2005) Deductive cause-consequence Analysis (DCCA). *In*: *The 16th IFAC World Congress*.

[23] Reer F., Reif W. (2007) Using deductive cause-consequence analysis (DCCA) with SCADE. *In*: *26th International Conference in Computer Safety, Reliability and Security*., pp.465-478.

[25] Güdemann M., Ortmeier F., Reif W. (2008) Computing ordered minimal critical sets. *In*: *Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems*.

[26] Lisagor O., McDermid J.A. (2006) Towards a practicable process for automated safety analysis. *In*: *24th International System Safety Conference*.

[27] Kulturel-Konak S., Smith A.E., Coit D.W. (2003) Efficiently solving the redundancy allocation problem using tabu search. *IIE Transactions*. **35**, pp.515-526.

[28] Kulturel-Konak S., Smith A.E., Norman B.A. (2006) Multi-objective tabu search using a multinomial probability mass function. *European Journal of Operational Research*. **169**, pp.918-931.

[29] Liang Y.C, Smith A.E. (2004) An ant colony optimisation algorithm for the redundancy allocation problem (RAP). *IEEE Transactions on Reliability*. **53**(3), pp.417-423.

[30] Zhao J., Liu Z., Dao M. (2007) Reliability optimisation using multiobjective ant colony system approaches. *Reliability Engineering and System Safety*. **92**, pp.109-120.

[31] Kim H., Bae C., Park S. (2004) Simulated annealing algorithm for redundancy optimization with multiple component choices. *In*: *Advanced Reliability Modelling, Proceedings of the 2004 Asian Internationals Workshop*. World Scientific, pp.237-244.

[32] Goldberg, D.E. (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, USA.

[33] Coit D.W., Smith A.E. (1996). Penalty guided genetic search for reliability design optimisation. *Computers and Industrial Engineering*. **30**(4), pp.895-904.

[34] Deb K., Pratap A., Agarwal S, Meyarivan T. (2002) A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*. **6**(2), pp.182-197.

[35] Konak A., Coit D.W., Smith A.E. (2006) Multi-objective optimisation using genetic algorithms: A tutorial. *Reliability Engineering and System Safety*. **91**, pp.992-1007.

[36] Papadopoulos Y.I., McDermid J.A. (1999) Hierarchically performed hazard origin and propagation studies. *In*: *18th International Conference in Computer Safety, Reliability and Security*. Toulouse, France, pp.139-152.

[37] ITI Gmbh. (2010) SimulationX 3 (www.simulationx.com)

[38] Rausand M, Oien K. (1996) The basic concepts of failure analysis. Reliability Engineering & System Safety, 53 (1996): 73-83.

[39] Parker D. & Y. Papadopoulos (2007), Effective Multicriteria Redundancy Allocation Via Model-Based Safety Analysis, IFAC Workshop on Intelligent Manufacturing Systems, 23-25 May, Alicante, Spain.

[40] Grante C. and J. Andersson (2003) Optimisation of Design Specifications for Mechatronic Systems, Research in Engineering Design 14(4):224-235.

[41] Sharvia S., Papadopoulos Y.I. (2008) Non-coherent modelling in compositional safety analysis. *In*: *IFAC 17th World Congress*. Seoul, South Korea.

[42] Coit D. W. & A.E. (1996), Reliability optimisation of series-parallel systems using a genetic algorithm, IEEE Transactions on Reliability R-45(2):254-260.

[43] Deb, K. (1999) Evolutionary Algorithms for Multi-Criterion Optimisation in Engineering Design, Evolutionary Algorithms in Engineering and Computer Science, 135-161. John Wiley & Sons.

[44] Grunske, L. (2006) Identifying ”good” architectural design alternatives with multi-objective optimisation strategies, 28th International Conference on Software Engineering (ICSE 2006), May 20-28, Shanghai, China.