TOWARDS INTEGRATED SAFETY ANALYSIS AND DESIGN

P Fenelon, J A McDermid, M Nicholson, D J Pumfrey

University of York

Abstract

There are currently many problems with the development and assessment of software intensive safety-critical systems. In this paper we describe the problems, and introduce a novel approach to their solution, based around goal-structuring concepts, which we believe will ameliorate some of the difficulties. We discuss the use of modified and new forms of safety assessment notations to provide evidence of safety, and the use of data derived from such notations as a means of providing quantified input into the design assessment process. We then show how the design assessment can be partially automated, and from this develop some ideas on how we might move from analytical to synthetic approaches, using safety criteria and evidence as a fitness function for comparing alternative automatically-generated designs.

Keywords: safety assessment, architectural design, goal structures, method integration, automated design

Introduction

Much current industrial practice in the design and assessment of safety-critical systems could, only slightly unfairly, be characterised as an 'over the wall' process. A design is produced with some cognisance of safety issues, it is 'tossed over the wall' to safety assessors who analyse the design and later 'toss it back' together with comments on the safety of the design and requests for change. Whilst something of a caricature, the above is not entirely unrepresentative of current industrial processes.

Industrial processes which have this character do so for organisational and cultural reasons — the design and safety departments are separate entities, populated by engineers with different skills — but there are also technical reasons. Specifically there is poor integration between safety analysis and design techniques, especially for software based systems, and the processes used do not easily accommodate the much tighter interaction between safety analysis and design needed for effective integration. In this paper we describe some of our work on producing more effective integration of safety analysis and design, and cover process issues, design and assessment methods and relate our work to more traditional safety analysis practices.

There are three major strands to our work. First, we propose a new way of organising and structuring development and assessment processes to encourage the stronger integration of design and analysis. Part of our aim in defining the process is to facilitate change management. For the purposes of this paper, however, we assume a 'top down' development model, but this should be viewed in the spirit of Parnas' 'rational design process — how and why to fake it' [28]. However we do recognise the need for investigating different designs, including assessing different design strategies for their safety properties.

Second, we consider adaptations of classical safety techniques to computer-based systems, introducing a modified form of the Hazard and Operability Study (HAZOP) for carrying out analysis of high level design proposals. We also show how techniques such as fault-trees and zonal hazard analysis can be adapted to software, and indicate how to carry out automated analysis of at least some safety properties, building on classical approaches such as Markov chains.

Third, we consider how to use the analysis techniques to guide design synthesis, i.e. deriving detailed designs from more abstract designs so that they have the desired safety and timing properties. Our approach uses heuristics for searching the design space, and the automated analysis techniques for 'pruning' the space, i.e. rejecting unsuitable designs. Whilst this work is in its early stages, it draws together the other two strands, and indicates the way in which we believe it will be possible to achieve a much more strongly integrated, and automated, design and safety analysis process.

First we discuss process issues, proposing a new way of modelling and controlling processes, and setting out the role of safety analysis in a design process. We use this discussion to set the rest of the paper in context.

Safety Analysis and its role in the Design Process

The design of a safety critical system inevitably involves trade-offs. Safety requirements may conflict with other requirements, e.g. for availability or performance, and compromises have to be found. The identification of conflicts between requirements, and their resolution, is therefore a central part of the design process; we have previously proposed the use of 'goal-structuring' as a way of making the 'spine' of the process clear [26], focusing on the derivation of requirements. It is our contention that these concepts help structure and document the complex processes of developing safety critical systems, particularly showing the relationship between safety analysis and design. We briefly introduce the concepts and show how they put the more detailed analyses discussed in the rest of the paper in context. The reader is directed towards surveys such as Leveson's [24] or Bennett's [3] for an overview of some of the techniques commonly used in software safety assessment, and to [4] for an overview of some of the classical means of achieving software dependability. An excellent survey of general dependability and reliability analysis methods is given in [35].

Goal Structuring Concepts

The two most fundamental concepts which are the basis of our process model are:

- goal — is something that someone wishes to be achieved; it is more general than a requirement and may encompass process issues (e.g. some action to be performed) and product issues, e.g. more conventional requirements;

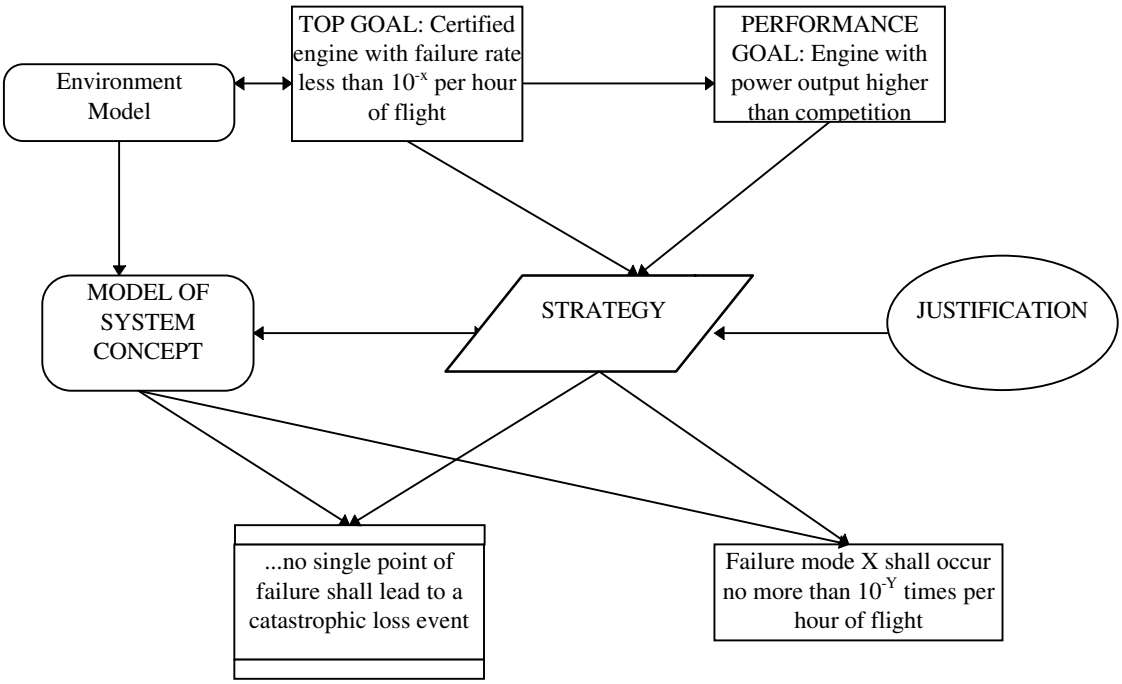

Figure 1: An Example Of A Goal Structure

- strategy — a strategy is a (putative) means of achieving a goal or set of goals, e.g. a system concept; a strategy will often generate sub-goals; meta-strategies can be used to represent the fact that a choice exists between two or more strategies.

Goals are decomposed through the strategies, and we will refer to sub-goals where this is helpful. Where there is a single root goal, the structure is a hierarchy. Where we have conflicting goals, or multiple goals which need to be satisfied at once, strategies will be introduced to resolve conflicts or to represent other forms of trade-off, and the structure will be a directed acyclic graph, as a strategy can satisfy more than one goal.

Some goals may be satisfied directly, e.g. by carrying out an action, or providing a product with the right properties. We use the term *solution* for the action or product which satisfies such a goal. Goals with solutions are *leaves* of the goal structure, i.e. they have no strategy or sub-goals.

We use the term *constraints* to refer to those goals which are not solved directly, but which restrict the way in which other goals are solved, i.e. which limit the set of allowable strategies (and models, see below). The satisfaction of constraints must be checked at multiple points in the goal hierarchy. Common safety requirements such as 'no single point of failure shall lead to a hazard' are representative of this class of goal.

It is intended that goals and strategies will provide the traceability in developing requirements, designs and safety cases. However there are other important facets of the structure which are not related to goals or strategies. These include:

- models — these represent part of the system of interest, its environment or the organisations associated with the system; goals will often be articulated in terms of models, especially when the model is an abstraction of the system design; models may be the (putative) solutions to goals, and will often be introduced by strategies (see below).
- justification — a justification is an argument, or other information, e.g. the results of a safety analysis, presented to explain why a strategy is believed to be effective; this may either justify the strategy at the time the choice is made, or retrospectively once solutions to the sub-goals have been provided.
- criteria — these are the basis for judging whether or not a goal has been satisfied; multiple criteria may be associated with a goal.

The standard simplifying assumption in any engineering endeavour is that we can have a 'top down' process, starting with the most abstract requirements, moving through layers of decomposition until the system is realised. We illustrate the above concepts by considering just such a 'simplistic' process involving the development of a safety case, before using the concepts to set the work of the rest of the paper in context. Treatment of 'real' processes such as issues of concurrent engineering is outside the scope of this paper.

An Illustration of the Process Model

To illustrate the use of the process model we describe a fragment of the process of decomposing requirements and developing a safety case — in parallel, based on the same goal

structure. We take the scope of our concern to be the development of a civil aero engine.

The top level goal will, in many cases, be to do with the licensing or certification of the equipment or system of interest. Thus, for an aero engine, the top level safety goal might be to 'certify the engine to the requirements of "JAR E", the Joint Aviation Regulation pertinent to civil aero-engines. Other high level goals will be concerned with performance, cost, etc. The first level of decomposition will typically introduce a system concept, i.e. a high level design and allocation of function amongst technologies, and the individual top-level safety (and other) requirements for the subsystems in the systems concept. The goals will then be to show that these requirements have been met[1]. Thus we might get goals such as 'show that failure mode X does not occur more frequently than 10^{-Y} times per flying hour'.

The example is illustrated in Figure 1 above. Two 'top level' goals are introduced — one concerned with performance, the other with safety. The goals may be articulated directly, or in terms of a model of the situation in which the engine will be used — the environment. A strategy for solving the goals is introduced which identifies a system model. This example introduces some direct safety goals, e.g. about allowable rates of failure modes. It also introduces a constraint (the box with two horizontal lines at top and bottom) which restricts the way that all the remaining goals are satisfied, i.e. it restricts lower level designs and strategies. In general the strategy will be constrained by the relevant standards, and the views of the certification and licensing body. In this specific example JAR E identifies safety targets for engines, including allowable failure rates for particular failure modes, and constrains the means of compliance with those requirements, including calling up standards and guidance such as DO 178 [30]. The justification will probably be by appeal to accepted practices or the requirements of the prevailing standards.

Decomposition of the safety goals will parallel the decomposition of the system design, and failure rates will be allocated to different components and their failure modes in such a way that satisfying the component level goals will satisfy the system level goals. This is carried on to the 'bottom level' of the design decomposition where the goals have to be satisfied by safety analysis, or other means of compliance. The role of the safety analyses is therefore to supply solutions to particular goals, and the justification for a strategy, e.g. a Markov analysis might be used to justify a strategy (design concept) by showing that the specified failure modes of components and the design of the system is such that the high level failure mode and rate targets are met. Where more than one strategy is proposed the safety analyses, and other evaluations, may be used to choose between alternatives. Thus the goal structure, the results of the safety analyses and aspects of the design information will form the basis of the safety case.

[1] The safety goals may not always be in terms of the requirements although they are in many cases, e.g. Civil Aircraft Certification. In some industries, the safety goals include showing conformance to standard designs; in this case the goals would not map directly to requirements.

Design and Analysis Process

The goal-structuring approach described above is very general, and capable of representing the relationships of designs, requirements and safety evidence. We need, however, to be clearer about the structure of the development process in terms of the information being manipulated, i.e. the contents of the models and the scope of the goals. We propose a simple phasing of the process:

- *requirements and system concept* — the system concept is the top level design for the complete engineered system, and the requirements are 'top level' goals and those derived requirements that must be met for the system concept to satisfy the top level requirements;
- *computing system architecture* — the structure of the computing application, including the processes and their inter-relationships together with their mappings to hardware;
- realisation — detailed design and program code.

We identify these phases in the process as the information and activities within one group are much more tightly coupled than they are between groups. Thus, for example, one would expect tight iteration between system concepts, requirements and associated safety analysis to arrive at a satisfactory set of requirements and associated system concept. This would then be a fairly stable input to the architectural grouping of activities. Of course there is feedback between the groups, but there will be less iteration, in an effective development process.

In this paper our attention is mainly focused on computing system architecture, although we will touch on the other parts of the process. This, in part, reflects our current areas of interest, and in part the desire to provide a reasonably complete treatment of some key issues, not a superficial treatment of the whole process.

Types of Analysis

To set the rest of the paper in context, we also need to consider the types of analysis that are used in the development process. Essentially all safety analysis techniques are concerned with establishing causal relations between failure modes, and those aspects of 'normal' behaviour which form a necessary part of the causal chain. It is common to distinguish top-down methods, such as Fault-Trees, from bottom-up methods, such as Failure Modes and Effects Analysis (FMEA). However we find that it is useful to make a finer distinction between analyses (as described further in [14]); we arrive at the taxonomy shown below (Figure 2).

	Known Cause	**Unknown Cause**
Known Effect	Description of behaviour	Deductive analyses
Unknown Effect	Inductive analyses	Exploratory Analysis

Figure 2: Taxonomy of Analyses

In our taxonomy the entries can be interpreted as follows:

- *known cause, unknown effects* corresponds to bottom-up analyses (e.g. FMEA);
- *unknown cause, known effects* corresponds to top-down analyses (e.g. FTA);

but:

- unknown causes, unknown effects, which we term *exploratory*, has no counterpart in the normal classification.

The exploratory approach is the form of analysis which is needed most in a 'design emergence' model, for example a new system concept will need to be investigated and we will know neither the causes nor the effects, and will be seeking to find them. Thus having effective exploratory safety analysis methods is central to our approach. We start our treatment of more detailed safety analysis issues with a consideration of exploratory safety analysis techniques, and their use for investigating alternative architectural designs. These represent the first stage in applying safety analysis to an emerging design and typically these analyses provide justification for high level strategies. We then discuss, in subsequent sections, adaptations of classical safety analyses for assessing detailed designs and implementations. Typically these analyses are used to confirm the soundness of designs, and thus provide solutions to some of the safety goals.

We next consider automation of some aspects of safety analysis, including addressing the need for trace-offs between different goals. We focus on timing as an additional type of goal, as many safety-critical systems are also real-time systems. Finally, we extend these ideas to illustrate how the analysis techniques may be adapted to support partial design synthesis.

Exploratory Safety Analysis

At the architectural design stage for new software, nothing is known about its failure modes, and knowledge of the effects of failure will generally be limited to a high level preliminary hazard analysis. However, this is the stage at which it is easiest and most cost effective to take measures to improve the safety characteristics of a system, so exploratory analysis is particularly important.

The SHARD (Software Hazard Analysis and Resolution in Design) technique described here was developed after we had conducted a survey of existing software safety analysis methods which concluded that no technique proposed so far adequately addressed the requirement for a structured exploratory safety analysis of a completely new software system. SHARD is based upon techniques derived from Hazard and Operability Studies (HAZOP) ([9], [22]), applied to software designs expressed in a structural notation such as MASCOT 3 [20] or DORIS [31]. Whilst the principles of our approach are general, we illustrate them in the context of MASCOT.

The primary intention of this analysis is to assess the safety (or otherwise) of the application software. The operating environment (e.g. the MASCOT run-time system) is assumed to be error free, an assumption which must be justified by other techniques such as formal verification and validation.

HAZOP, a structured system of imaginative anticipation of hazards and the suggestion of means of overcoming them, was developed by ICI in the mid 1960s to study the design of new chemical plant. Various recent papers ([7], [8] and [12]) have suggested adaptations of HAZOP to the software environment, but the SHARD method is distinct from these in the emphasis it places on using the results of the analysis to drive design improvements.

Key features of the HAZOP technique adopted for SHARD are:

- The analysis is based on consideration of the properties and behaviour of flows. In a chemical plant, these are the pipelines connecting components such as pumps and reactor vessels; in software, they are the control and data flows between processing components.
- The analysis uses a set of guide words to suggest hypothetical failure modes for consideration.

The capability of the method to identify all the important failure modes of a system clearly depends upon the selection of appropriate guide words. The set of guide words used in the process industries HAZOP method has been developed and refined by the experience of many years' application of the method; for application to software, it is necessary to propose a means of developing a set (or sets) of guide words with a high degree of confidence in their completeness.

Guide word selection in the SHARD method is based upon the considerable body of research work (e.g. [13] and [5]) in the field of software failure classification.

From this work, the following failure classes have been identified as the basis for guide word generation in SHARD:

Service provision : Omission, Commission

Service timing : Early, late

Service value : Coarse Incorrect, Subtle Incorrect

For each specific application, these basic failure classes are refined by considering their interpretation in the context of a particular combination of data type and path protocols (i.e. the communication model, which determines the timing and service provision characteristics of a flow). For some combinations of data type and path protocol, consideration of one fault class may result in the definition of more than one guide word. The guide words thus defined are recorded in a table, and the appropriate set selected as each flow is analysed. Figure 3 shows a typical table of guide words derived for some basic combinations of MASCOT data types and path protocols.[2]

The starting point for analysis of a software design is the top-level (context) MASCOT diagram of the system, *i.e.* its logical architecture, developed from the functional requirements of the system. The information (control and data) flows in the diagram are identified and consistently labelled, and the design is reviewed to ensure that the intended behaviour of each flow under normal operation is clear. The appropriate set of guide

[2] Pools are shared data areas with destructive write. Channels are buffered streams with destructive read..

		Failure Categorisation					
Flow		**Provision**		**Timing**		**Value**	
Protocol	**Type**	*Omission*	*Commission*	*Early*	*Late*	*Subtle*	*Coarse*
Pool	Boolean	No update	Unwanted Update	N/A	Old Data	Stuck at...	N/A
	Value	"	"	"	"	wrong in tolerance	out of tolerance
	Complex	"	"	"	"	Incorrect	Inconsistent
Channel	Boolean	No Data	Extra Data	Early	Late	Stuck at...	N/A
	Value	"	"	"	"	wrong in tolerance	out of tolerance
	Complex	"	"	"	"	incorrect	inconsistent

Figure 3: SHARD Guidewords for MASCOT

words for each flow is then selected by reference to its path protocol and data type.

Taking a flow at a time, each guide word is considered in turn, and may suggest one or more hypothetical failure modes, which are recorded. For each hypothetical failure mode, possible causes and potential consequences are sought — effectively introducing both deductive and inductive phases into the analysis. If a hypothetical failure can be shown to have both conceivable cause(s) and hazardous consequences, it is termed a meaningful failure mode, and consideration must be given to measures which can be taken to remove its causes or limit its effects. An important feature of the method is that, for those hypothetical failure modes which are not considered meaningful, a justification must be given for this decision. In most cases this will be a simple statement but, where the decision is difficult, it may be necessary to supply a more complete argument, perhaps based on design refinement and the application of more detailed safety analysis.

For each meaningful failure mode identified, alternative strategies are suggested for removing its causes or limiting its effects. These may include proposing alternative designs, modifying the current design, or establishing further requirements which must be satisfied by lower-level design elaboration to achieve acceptable system-level safety properties. These alternative strategies are evaluated either by repeating the analysis for revised design proposals, or by proposing and analysing designs for the next level of decomposition, taking account of any new derived requirements. As the design is progressively refined, SHARD will not be able to support the detailed analysis of issues such as schedulability and resource usage which are required to select between alternatives, and techniques for automating analysis must be applied.

The results of the SHARD analysis are recorded in a tabular format similar to that used for FMEA), which also includes fields for recording the alternative strategies considered and a justification for the eventual selection, although these may simply contain pointers to other documents.

We can now relate the use of SHARD to the concepts introduced earlier. A meta-strategy will represent the evaluation of design alternatives. Each strategy will have an associated model, defining the design (the MASCOT diagram) and a subgoal to carry out the SHARD analysis. The SHARD tables will be the solution to the subgoal and the derived requirements will form further subgoals. The justification for classifying particular failure modes as non-meaningful will be part of the justification of the strategy. The justification for the meta-strategy (selecting the "best" strategy will be a comparative evaluation of the safety properties (including failure modes) of the designs, and other relevant information, e.g. cost.

Safety Analysis Notations

Various causally based techniques for assessing systems safety based in known designs have evolved. These traditionally fall into two classes — methods which work from known causes to unknown effects (such as Failure Modes and Effects Analysis) [1] or those which work from known effects back to initially unknown causes (such as Fault Tree Analysis) [34]. Unlike SHARD, these techniques may only be used once we have a fully detailed design, although fault trees can easily be applied to incomplete designs.

In classical safety engineering applications, these techniques are probabilistic and quantitative. In the software domain, we must treat them as deterministic and quantitative, but when considering the interaction with the rest of the system (actuators, sensors etc.) we may then need to reconsider probabilistic aspects of the system's behaviour and integrate data from software safety analysis with that derived from probabilistic assessment of other parts of the system.

Fault Trees

Classical fault tree analysis (FTA) is formulated in terms of a recursive causal ordering of events which contribute to a given undesired top event (i.e. a hazardous failure mode); AND and OR combinators (and simple variations on these) are used to structure the event space, and analysis techniques allow the reduction of fault trees to standard sum-of-products ("cutset") forms. In fact, AND is the only truly causal combinator in fault tree analysis; OR-decomposition merely represents alternative event sequences with the same net causal effect. The probability

of the top event may be calculated if those of the leaf events are known — as the sum of products of the probabilities of the tree arranged in minimal cutset form.

Various applications of FTA techniques to software have been made. Taylor's cause-consequence analysis influenced work on software FTA [33] and Leveson's template-based approach to FTA for Ada programs [23] provide satisfactory descriptions of the failure behaviour of small fragments of program code, but the large-scale structure of the system is not reflected in the fault trees generated. In particular, Leveson's approach based around the use of instances of template fault trees for individual program statements composed according to a set of rules gives rise to fault-trees constructed from the bottom up. In fact, it can be thought of as a form of Failure Modes and Effects Analysis using FTA notation.

Our initial approach [14] merged some of the low-level structuring offered by Leveson's system — the templates — with traditional top-down construction of fault trees. We partitioned programs into related groups of statements — effectively, we are carrying out a manual form of decomposition into components about which we can reason independently.

Later developments to our method exploited the observation that most safety-critical programs contain well-defined fragments of code whose higher-level failure modes we already understand in terms of their failure behaviour at the systems level — for example, assignment of default values to out-of-range readings in data acquisition yields subtle incorrect but timely failure modes, and so on. Such programming clichés may be identified by reference to a pre-defined library of software components (this gives us the theoretical power to re-use safety analysis data in software development) or may be discovered on an application-specific basis — in practice, a mixture of these approaches is usually necessary. A simple fault tree — OR'ing together the basic failure modes of the component — replaces the template-based trees which would otherwise have been generated from the statements. In many cases, this approach means that fault tree analysis down to the code level is not necessary, assuming we have sufficient understanding of the components being re-used.

Rates of individual failure modes can be combined to give failure rates for effects, and propagated through FTAs to give the rate, or probability, of the hazardous “top event”.

Failure Modes And Effects Analysis

The top-down, component-structured approach to fault tree analysis we propose integrates well with Failure Modes and Effects Analysis (FMEA), a complementary technique which is often used with FTA in industry. FMEA works from known component failure modes and rates (in many industries, component suppliers provide tabular details of failure modes and probabilities to systems integrators) to unknown system-level effects; it is essentially a tabular, labour-intensive process with a highly experiential focus.

FMEA has had relatively few successful applications to software to date. For example, Raheja [29] has proposed a technique which is clearly a derivation of classical FMEA, but does not appear to capture the essence of a software “component” in a well-defined way.

We believe that our FTA structures (particularly the cliché-identification) will let us provide data which will integrate well with FMEA-style analyses of the underlying hardware infrastructure and related sensors and actuators; hardware failure modes can provide input to software fault trees, and the output failure modes of software components can be fed into higher-level FMEAs and FTAs of the overall system.

Failure Propagation And Transformation Notation

We have developed an integrated notation to ameliorate some of the limitations of FTA and FMEA when applied to software. Fault trees, even those formed by our hierarchical hybrid approach, often tend to be intractably large. Also, although FMECA is a powerful technique for analysing well-known failure modes its textual worksheet format makes it difficult to trace effects from one level to another.

What is needed is a graphical notation which allows us to work in both the top-down mode offered by FTA and the bottom-up mode of FMECA. Ideally this notation should be compact and should enable “end-to-end” modelling of the failure behaviour of a system. We believe that our Failure Propagation and Transformation Notation (FPTN) offers the desired properties. The notation is quite general — it is not inherently specific to software or even to computing systems in general — but fits in well with the MASCOT and SHARD methods.

Basics

In FPTN a system is represented as a number of *modules* each corresponding to some functional element within that system. Modules have standardised *attributes*; a name, a criticality level and (where applicable) indication of any recovery mechanisms which might exist in that module.

Modules may be nested hierarchically, and can be either “black box” (non-decomposable modules at the lowest level of abstraction being considered) or “white box” (decomposable with a known internal structure). Example FPTN modules appear in figures 4 and 5.

In FPTN modules are connected not by the data which normally flows between them, but by the failures which propagate between them. Inside each module we list those failure modes which are generated purely internally by that module (i.e. do not depend upon any external conditions), those failure modes which are unconditionally prevented from propagating further by the module (failures caught by exception handlers or other recovery mechanisms) and also a set of equations characterising the relationship between *input* and *output* failure modes, where input failure modes are those to which the component is susceptible and output failure modes are those which it passes on to its environment. Again we can see the relevance of the failure mode classifications: we would expect to detect coarse failures but not subtle ones, and so on.

These equations take the form of a set of logical relationships between inputs and outputs — each equation describes the relationship between some subset of the input failure modes and one of the output failures. The canonical form of these equations is a “sum of products” form which is isomorphic to the minimal cutset form of a fault tree, with the output failure mode being considered as the top event. Thus each FPTN module

corresponds to an abstraction of a set of fault trees describing a particular component.

So far we have only described failure *propagation*. Where does *transformation* come into the notation?

We can consider failures as lying in particular *domains* such as those used as guidewords in figure 3. We would expect users of FPTN to add or substitute failure modes appropriate to their application domain as necessary — for example, the clichés we referred to above will often introduce new types of application-specific failure mode.

In many systems the domain of a failure may be changed as it propagates through the system. For example, some computation in a real-time computer system may overrun its budgeted execution time, thereby triggering a watchdog timer which forces it to return an approximate result — in this example a time domain failure (the overrun) has been transformed into a (hopefully subtle) value-domain failure (the approximate result). Such structures occur so widely in large software systems that our notation has been tailored to model them. Each failure mode in a system is typed with its domain and in addition to describing the logical relationships between failure modes the type annotations allow FPTN to describe the changes in failure domain which take place.

Scope and Usage of FPTN

The flexibility of FPTN means that it has many potential uses, of which this paper describes several. At its most basic level FPTN can act as an abstraction and representation of complex fault trees and/or FMECA tables; as we become more ambitious we can use FPTN to bring many other systems safety activities into the software domain because of its rich causal model. The notation can be tied closely to the design process and, at the highest levels of abstraction, can be used as a system model in high-level argumentation about system safety, as input to a tool such as SAM [15].

FPTN can have two roles as part of the overall safety analysis. Most obviously, it can be used as (part of) the safety analysis of some artefact. Thus it might be used directly to provide the solution to some leaf goal. More plausibly, it might be used to provide a summary of more detailed FTA and FMEAs which is more meaningful in terms of overall system behaviour.

Less obviously, but perhaps more usefully, FPTN can be used to specify failure behaviour requirements — to state what output failure modes are acceptable, given the assumed input failure modes. Thus FPTN might be used to state a derived requirement (subgoal) arising out of a SHARD analysis, for the allowable failure behaviour of a design component. The solution to this goal might well be an FPTN model derived from the *actual* artefact.

An FPTN "solution" satisfies an FPTN "goal" if the failure modes and rates are "no worse", *i.e.* there are no additional hazardous failure modes and the rates are less than or equal to those in the goal. The failure rates and probabilities are propagated through the solution FPTN in the same way as they are through fault-trees.

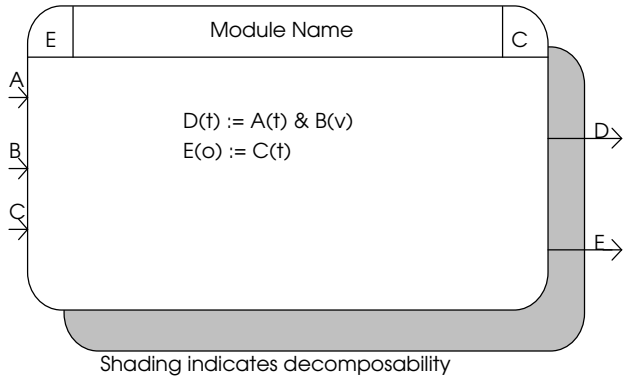

Figure 4: An Example Of An FPTN Module

Zonal Analysis

FTA, FMEA and SHARD are based on the *logical* structure of the design. However, in *systems* safety assessment a number of experiential analyses based upon knowledge of the *physical* structure of the system and arrangement of its components are commonly carried out. Zonal Hazard Analysis (ZHA) is typical of these processes; in its usual aerospace domain ZHA considers the interactions of logically unrelated systems in the same physical part (zone) of an aircraft (e.g. nose, wings, etc.). For example, ZHA would consider the effect of a hydraulic leak on electrical connectors in the same zone. Similar approaches have not, as far as we are aware, been applied to software systems.

In part this is due to the failure, mentioned above, of many approaches to software FTA and FMECA to correctly identify the software components and their failure modes. Also, in attempting to carry out a ZHA-like process on software we need to consider the tricky problem of what actually constitutes a "software zone".

FPTN can assist in the identification and modelling of zones. We assume a typical computational model as found in many real-time and safety-critical systems — one in which a number of communicating tasks are mapped on to a set of processors, communicating either by shared memory or message passing over a bus. Each task is identified with an FPTN module. We note that the failure propagation will not normally respect the logical structure of the design; for example, it may propagate via memory corruption.

We can define a zone in several ways appropriate for different kinds of analysis, and different 'strengths of protection' of the run-time system and infrastructure:

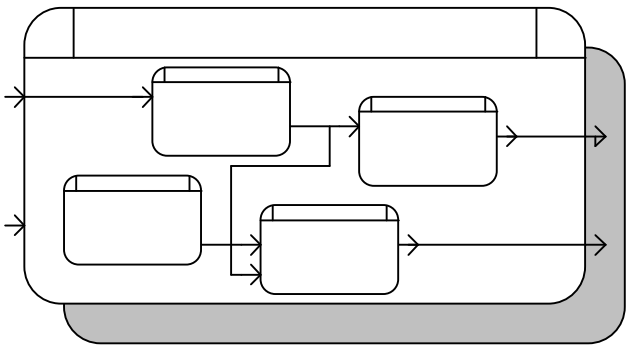

Figure 5: Example Of A Composite FPTN Module

- **"Strict" zones:** we consider all tasks in the system and all propagations of failures between them. If we can partition the system task set into two or more disjoint subsets (zones) such that no failures propagate between the subsets (of course we must consider the *transitive closure* of the propagation) then these can be termed *strict zones*
- **"Task-based" zones:** for each task (module) we can identify a set of other tasks which are affected by the transitive closure of failure modes propagated from it. We can therefore consider in general terms the parts of a system "infected" by arbitrary failures of a particular task or module.
- **"Failure-based" zones:** are simply defined as the transitive closure of modules affected by a *particular* failure mode.

For ZHA-like partitioning of a system to be accurate we need to add such failure domains as "addressing error", "communications failure" and so on, as mentioned in the section on FPTN to reflect the impact of infrastructure failures on the application. Where the kernel, or run-time system, implements segregation domains we would need to model all failures as passing through the kernel and, and to represent it in FPTN.

Several interesting properties arise from this new form of analysis; we can reason about *graceful* (or otherwise!) *degradation* of a system by deducing what zones are affected by particular failures; we can use our knowledge of zones within the system to place redundant or replica tasks such that common-mode failures do not affect them and we can provide input into the task allocation problem. Indeed software ZHA can perhaps best be thought of as a form of common-cause analysis.

This use of FPTN to perform zonal analysis may provide a solution to a leaf goal. Specifically, it may be used to show that the constraint "no single point failure shall lead to a hazard" is met, even considering the effects of common-mode software failures on the design. However it is unclear how valuable this possibility is as, in many cases, protection against single point failure would be provided by hardware redundancy.

Mechanising Safety Analysis

SHARD and FPTN, together with more classical techniques such as FTA and FMEA, enable the failure behaviour of systems to be described logically. They also facilitate numerical analysis of failure rates and probabilities. Logical analysis is useful for selecting between outline designs early in the process, but the numerical data is needed to "close out" the analysis — to provide the solutions at the bottom of the goal structures. We consider the numerical analysis of safety properties in the context of Simpson's Data Interaction Architecture (DIA) [31], which supports point-to-point inter-node communication.

As mentioned earlier, safety-critical systems are also often real-time systems. Thus we need to ensure that a design satisfies its timing goals as well as its safety goals. We first consider timing issues.

Analysis of an Allocation

Consider the problem of analysing a given allocation of application tasks to processors. The information required includes the set of processors, which processors are directly linked, and which tasks reside on which processors. The task set includes intermediaries (routing tasks) for those processors that are not directly linked. Also, the worst case execution times (WCETs) of tasks are required. A number of tools now exist to estimate the worst case execution time of a task [36], [16]. At higher levels of design, time budgets can be used in place of calculated WCETs. These are then replaced with actual figures as tasks are instantiated as code. The message traffic between tasks also need to be known. In hard real-time systems transactions, *i.e.* sets of precedence constrained tasks, have response deadlines placed upon them. The transactions in the system need to be identified.

If the tasks in a transaction are placed on a processor, or set of processors that no other tasks in the system reside on, the worst case response times produced for the subsystem tasks can be summed to give the system response time. However, if other tasks can reside on the same processors then the worst case timing characteristics of the transaction are subject to interference from higher priority tasks and blocking from lower level tasks from other transactions.

Analysis of the allocation requires satisfaction of simultaneous equations identifying the interactions between the tasks. Solution of these equations can be based on unique priority static scheduling of hard real-time systems. Audsley *et al* [2] introduce the appropriate scheduling theory and Burns *et al* [6] apply the theory to DIA.

For an allocation to be feasible, not only must all the tasks and transactions meet their deadlines but the available resources must not be exceeded. This forms the set of criteria for the satisfaction of a timing/resource usage goal. For our exemplar architecture, resource constraints include the maximum number of tasks allowable on a processor, the maximum private and shared memory capacity of processors and links respectively, and the maximum number of messages that can be sent down each link. An allocation can easily be checked to show that it is feasible with respect to its resource constraints.

Choosing an Allocation

Given that a particular allocation can be analysed for its timing and resource usage how can we choose between the set of possible allocations? This can be done manually, but it can also be automated, using the scheduling test as an acceptance criterion. This is a combinatorial optimisation problem in which a solution space needs to be traversed to find the best feasible solution. A number of search techniques exist, which are variants of the neighbourhood search paradigm. Simulated Annealing [21] which considers a set of single solution points has proved to be an effective method of solving the allocation problem for known hardware configurations of DIA. Genetic Algorithms [20] allow a population of solutions to be used and should allow extensions to allow the configuration and safety design elements to be considered. (see below). In the Simulated Annealing (SA) approach a single initial, usually random, solution is produced and evaluated against a 'goodness' value which forms the criterion for choosing between alternatives. For the exemplar architecture with known tasks placed on given hardware elements the following 'goodness' criterion is used for timing properties

$$E_p = K_0 E_{mem} + K_1 E_{dual} + K_2 E_{dead} + K_3 E_{task} + K_4 E_{time}$$

where

K_i is a weighting factor for element i

E_p = 'goodness' criterion for proposed allocation p

E_{mem} = excess private memory required over capacity available. 0 if less than maximum used.

E_{dual} = excess shared memory required over capacity available. 0 if less than maximum used.

E_{dead} = penalty associated with the number of tasks that cannot meet their deadline in the worst case.

E_{time} = penalty associated with the total task response time of the system

E_{task} = penalty associated with the number of routing tasks in the system (for forwarding messages between processors not connected by a point-to-point link).

The algorithm will seek feasible solutions, then minimise the value associated with this measure. The solution is altered either by changing the priority of a task or moving a set of tasks to different processors. This new solution is evaluated and if it is better, *i*.e. has a smaller value than the previous solution, it is accepted. In order to reduce the chances of converging to a local minimum a poorer solution can also be accepted. The probability of accepting such moves is reduced as the algorithm progresses. Thus we can think of the optimisation technique as providing a meta-strategy which generates and evaluates many strategies, selecting a near-optimal result.

Safety and the Allocation Problem

The use of safety design idioms will have an effect on the timing and resource usage aspects of a design. The allocation of tasks may be constrained by the idiom being used. To show the effect of safety design idioms we will consider the application of watchdogs and replication techniques to our exemplar architecture.

Watchdogs can help guard against timing failures in architectures, such as DIA. A table of worst case execution times can be produced from analysis of the code of a task and the watchdog set to this value. Tasks which over-run their times can be made to fail silently or produce a known erroneous value, *i.e.* to turn a timing failure into a subtle value failure as mentioned above. The impact of watchdogs on the allocation problem is to increase the worst case execution time of the task. If the watchdog interrupts the task it may cause extra code to be executed. This must also be included as part of the worst case execution time of the task.

Replication of a task or more importantly a transaction, has a greater impact on the allocation problem. A number of different replication protocols can be envisaged to help mask the effects of permanent or transient faults. Most forms involve the production of a second set of task(s) which are not allowed to be on the same processor(s) or use the same link(s) for message passing, as the original(s). This is particularly difficult in DIA because of the use of routing tasks and the temporal decoupling of the processors in a DIA network. [32]

One or more decision tasks are often required to decide which value is to be propagated through the system. Thus if replication is to be used we need to identify the transactions to which it is to be applied, and the form it is to take.

The goodness value of a given allocation needs to indicate whether the proposed allocation of replica tasks or use of watchdogs is feasible. In addition it must also take into account the failure properties, specifically the failure modes propagated through the system. This can be evaluated by combining the FPTN models for the code modules, and computing rates of failure modes (if the basic data from hardware FMEAs is available). The FPTN goals form criteria for acceptability of the allocation. However, the failure rate computation is complex.

A number of simple reliability measures, such as reliability block diagrams, are deterministic. However most realistic strategies do not admit deterministic solutions. Typically Markov and semi-Markov analysis is required and these techniques are very computationally expensive. Powerful tools to estimate and predict system reliability have been produced. The main comparative work on these tools considers five tools [17].

Zonal analysis and analysis reuse should allow reduction of the (re)computation required to produce new failure rate estimates for designs. The HARP [11] tool, for instance, has a number of features that help integrate the tool with the other analytical methods discussed in this paper. HARP allows the fault model to be input in the form of a fault tree (recall that FPTN models can be viewed as sets of fault trees). It then automatically transforms the tree into an appropriate Markov chain. To model sequence dependencies it provides four new fault tree gates. Fault tree analysis also allows logical analysis to be carried out to go with the quantitative analysis [10].

Thus, in principle, part of the design process can be automated. In effect, the optimisation techniques form meta-strategies and the timing and failure criteria can be used to select acceptable designs. We have demonstrated this approach on small examples in the SPIRITS project, but there are many technical problems to address.

Design Synthesis

We have considered how to produce and evaluate alternative designs for timing and safety properties. We have demonstrated how, for a given architecture, with known hardware components (links and processors), and a proposed set of safety design idioms (to mask or remove failure modes) tasks can be allocated to processors such that no safety, timing or resource constraints are violated. The safety of the proposed design is measured by the failure modes and rates. Our design synthesis approach based on Simulated Annealing can address this problem, but Genetic Algorithms seem to be a better prospect for developing effective automated design tools.

Genetic Algorithms are population based heuristic optimisation techniques that are used to navigate a solution space. The solution space to be searched is the safety, timing, resource usage and functionality characteristics of alternative designs. A Genetic Algorithm can be characterised as a two stage process. It starts with a current population. Selection is applied to this population to create an intermediate population. Then recombination and mutation are applied to the intermediate population to create the next population.

For the allocation problem already considered using Simulated Annealing, a variant can be produced. The full Chromosome consists of appropriate genes of processor, priority and route chromosomes. The fitness function for any given possible solution is analogous to the goodness criterion in SA. In this approach the selection mechanism to determine members of the population to survive to the next generation is a Boltzmann tournament [25].

The GA approach can be extended to incorporate other aspects of a design. For instance, it can consider different infrastructures. For our exemplar DIA the number of processor and links in a system may not fixed. The configuration of links can also vary. The chromosome is extended to include two new chromosomes describing the liveness of each processor and link, and the fitness function can be extended to accommodate the extra effects of changing the infrastructure.

A similar approach can be adopted to consider the use of different safety design idioms that could potentially overcome identified failure modes in system components, both hardware and software.

From our limited experiments, it has become apparent that the designer can dramatically affect not only the speed with which any algorithm reaches a solution but also the quality of the solution produced. For instance some allocations are known to be infeasible because tasks need to be connected to particular peripherals, or need to be placed on a particular type of processor. A neighbourhood search technique, both simulated annealing and genetic algorithms are neighbourhood techniques, called Tabu search [18] could provide a method of including designer information. In Tabu search, a dynamic set of user-defined rules, which defines those neighbours that are tabu, is produced. Tabu search is still in its infancy, and is being expanded to include Aspiration Criteria, where rules are produced to indicate that certain moves are preferred over others. Each rule can be given a weight, negative if tabu, and positive if an aspiration. This would appear to offer a direct way of evolving criteria in a design synthesis environment, based on our approach to goal structuring,

Conclusions

We have described some adaptations of classical hazard and safety analysis techniques to handle software intensive systems. We have also considered the possibilities of automating design analysis and synthesis, taking into account the need to satisfy multiple goals simultaneously. The techniques we have discussed fit into the overall process as illustrated in figure 6, where we have elided the strategies to show the relationship of the design to the top-level goals.

The techniques we have described can be applied manually, or with simple tool support to modest-sized designs. We have applied SHARD, FPTN and our hierarchical software FTAs to a number of case studies derived from real systems. We have found the techniques practical and have, in some cases, found significant flaws in designs, e.g. failure modes that the designers had overlooked. Our ideas on software ZHA remain hypothetical, although we believe that they will prove relevant as safety-critical systems become more complex.

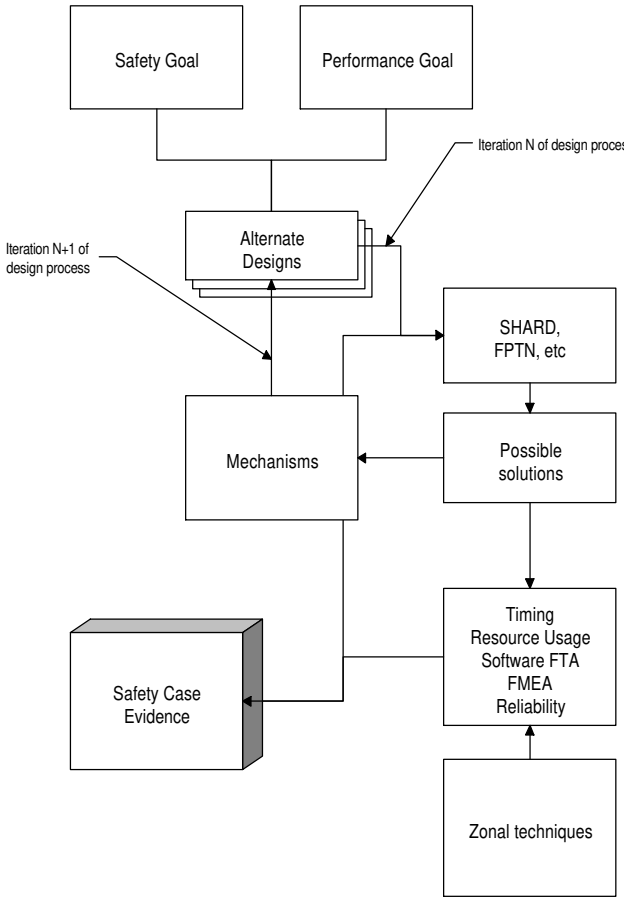

Figure 6: A Model Of The Design Process

We have also applied the design analysis and synthesis approach to medium-scale applications consisting of a few tens of tasks on 16 processors as part of the SPIRITS project. The approach based on simulated annealing works reasonably well when dealing with timing alone, but becomes very computationally expensive when addressing safety, especially vastly different design idioms. More effective techniques such as genetic algorithms, are needed if this approach is to prove practical.

Other groups have carried out work on adapting safety analysis techniques to software, and the work of Leveson and Taylor is of particular historical significance. However, we are not aware of any other work which has provided such an extensive set of safety analysis techniques for software based systems, nor made so much progress in showing how to integrate them into the design process.

Whilst much remains to be done, we believe that we have identified a fruitful line of research and we aim to use the Safety Argument Manager (SAM) [15] currently under development to draw together these strands of work, with the aim of further proving their effectiveness.

Acknowledgements

The authors would like to thank their colleagues (notably Ken Tindell, Neil Audsley, Rob Davis, Alan Burns and Andy Wellings) in the Real Time Systems research group whose work provided the underlying theory for the task allocations discussed earlier in the paper. Stephen Wilson of the ASAM-II project has also provided many useful comments and much advice.

Much of the work described in this paper has originated on the SSAP (Software Safety Assessment Procedures), ASAM-II (A Safety Argument Manager), SPIRITS and DCSC (Dependable Computing Systems Centre) projects at York and we extend our thanks to our industrial partners in these projects, most notably various operating companies within British Aerospace.

Contact Addresses

The authors can be contacted at the following address:

High Integrity Systems Engineering Group,
Department of Computer Science,
University of York,
Heslington,
York,
Y01 5DD,
United Kingdom.

Their electronic mail addresses are `pete` (Peter Fenelon), `jam` (John McDermid), `mark` (Mark Nicholson) and `djp` (David Pumfrey) `@minster.york.ac.uk`.

References

[1] ARP926, "*Design Analysis Procedure For Failure Modes, Effects and Criticality Analysis (FMECA),*" Society of Automotive Engineers (SAE), Detroit, USA (15th September 1967).

[2] Audsley, N., A. Burns, M. Richardson, K. Tindell and A.J. Wellings, "*Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling,*" Software Enginnering Journal (Sept 1993) pp.284-292.

[3] Bennett, P. A., "Safety," in Software Engineer's Reference Book, ed. John A. McDermid (1991).

[4] Bishop, P. G., "Dependability Of Critical Computer Systems 3," Elsevier Applied Science (1990).

[5] Bondavalli, A. and L. Simoncini, "*Failure Classification With Respect To Detection,*" in Volume 2: First Year Report, Task B: Specification and Design for Dependability, ESPRIT BRA Project 3092: Predictably Dependable Computer Systems (1990).

[6] Burns, A., M. Nicholson, K. Tindell and N. Zhang, *"Allocating and Scheduling Hard Real-Time Tasks On A Point-To-Point Distributed System",* in Proceedings of the Workshop On Parallel and Distributed Real Time Systems, (April 13-15 1993).

[7] Burns, D. J., and R M Pitblado, "*A Modified Hazop Methodology For Safety Critical System Assessment,*" in Proceedings of the First Safety Critical Systems Symposium, Springer-Verlag (1993).

[8] Chudleigh, M., "*Hazard Analysis Using Hazop: A Case Study,*", pp.99-108 in Safecomp 93: Proceedings of the 12th International Conference On Computer Safety, Reliability and Security, ed. J. Gorski (1993).

[9] CISHEC, "*A Guide To Hazard And Operability Studies,*" The Chemical Industry Safety and Health Council of the Chemical Industries Association Ltd. (1977).

[10] Dugan, J. B. and M. A. Boyd, "*Fault Trees & Markov Models for Reliability Analysis of Fault-Tolerant Digital Systems,*" Reliability Engineering & System Safety 39 (1993) pp.291-307.

[11] Dugan, J. B., S.J. Bavuso and M.A. Boyd. *"Modelling Advanced Fault-Tolerant Systems With HARP"*, 1991 Annual Reliability and Maintainability Symposium (1991).

[12] Earthy, J. V., "*Hazard and Operability Study As An Approach To Software Safety Assessment,*" Proceedings of the IEE Colloquium on Hazard Analysis (11 November 1992) pp.5-1 em 5-3.

[13] Ezhilchelvan, P. D. and S. K. Shrivastava, "*A Characterisation of Faults in Systems,*" Technical Report 206, University of Newcastle upon Tyne, Computing Laboratory (Sept 85).

[14] Fenelon, P. and J A McDermid, "*An Integrated Toolset For Software Safety Analysis,*" Journal Of Systems and Software (July 1993).

[15] Forder, J., C. P. Higgins, J. A. McDermid and G. Storrs, "*SAM — A Tool To Support The Construction, Review and Evolution Of Safety Arguments,*" in Directions In Safety-Critical Systems: Proceedings Of The Safety-Critical Systems Symposium, Bristol 1993, ed. F. Redmill and T. Anderson, Springer-Verlag (1993).

[16] Forsyth, C. H., "*Implementation of the Worst-Case Execution Analyser,*" Hard Real-time Operating System Kernel Study Task 8, volume E, York Software Engineering Ltd., UK (1992)

[17] Geist. R. and K. Trivedi, "*Reliability Estimation of Fault-tolerant Systems: Tools and Techniques,*" Computer 23(7) (July 1990) pp.52-61.

[18] Glover, F., E. Taillard and D. deWarra, "*A User's Guide to Tabu Search,*" Annals of OR 41 (1993) pp.3-28.

[20] Goldberg, D. E., "*Genetic Algorithms in Search, Optimization and Machine Learning,*" Addison-Wesley (1989).

[20] JIMCOM, "*The Official Handbook of Mascot,*" Joint IECCA and MUF Committee on Mascot (1977).

[21] Kirkpatrick, S., C. D. Gelatt and M. P. Vecchi, "*Optimization By Simulated Annealing,*" Science 220 (1983) pp.671-680.

[22] Kletz, T., "*HAZOP and HAZAN: Identifying and Assessing Process Industry Standards (3rd Edition),*" Institution of Chemical Engineers (1992).

[23] Leveson, N. G. and P.R. Harvey, "*Analyzing software safety,*" IEEE Transactions on Software Engineering vol.SE-9, no.5 (Sept. 1983) pp.569-79. IEEE Trans. Softw. Eng. (USA).

[24] Leveson, N. G., "*Software Safety - What, Why And How?,*" ACM Computing Surveys 16(**2**) (June 1986) pp.125-164.

[25] Mahfoud, S. W. and D. E. Goldberg, *"Parallel Recombinative Simulated Annealing: A Genetic Algorithm",*ILLiGAL report 92002 (1993).

[26] McDermid, J. A., A. Coombes and P. Morris, "*Causality as a means for the expression of requirements for safety critical systems,*" in Compass '94 (1994 to appear).

[27] Nicholson, M, T. Manning, A. Burns, M. Nicholson, K. Tindell and N. Zhang, "*Safety and Failure Modes Analysis — Allocating and Scheduling Hard Real-time Tasks on a Point-to-Point Distributed System*", Proceedings of the Workshop on Parallel and Distributed Real-Time Systems (April 13-15 1993).

[28] Parnas, D. L. and P. C. Clements, "*A Rational Design Process: How And Why To Fake It,*" IEEE Transactions On Software Engineering SE-12 (1986).

[29] Raheja, D., "*Software System Failure Mode And Effects Analysis (SSFMEA) — A Tool For Reliability Growth,*" Proceedings of the International Symposium on Reliability and Maintainability (1990) pp.IX-1 - IX-7.

[30] RTCA, "*Software Considerations in Airborne Systems and Equipment,*" Document No. RTCA/DO178A, Radio Technical Commission For Aeronautics, USA (1984).

[31] Simpson, H. R., *"A Data Interaction Architecture (DIA) for Real-Time Embedded Multi-Processor Systems"* Computing Techniques In Guided Flight conference, RAe (19 April 1990).

[32] Simpson, H. R., "*Four-slot Fully Asynchronous Communication Mechanism,*" IEE Proceedings on Computers and Digital Techniques 137 (January 1990) pp.17-30.

[33] Taylor, J. R., "*Fault Tree and Cause Consequence Analysis for Control Software Validation,*" Riso-M-2326, Riso National Laboratory, DK-4000 Roskilde, Denmark (Jan 1982).

[34] Veseley, W. E., "*Fault Tree Handbook,*" Division of the System Safety Office of Nuclear Reactor Regulation, US Nuclear Regulatory Commission, Washington DC (1981).

[35] Villemeur, A.,*"Reliability, Availability, Maintainability and Safety Assessment: Volume 1: Methods And Techniques"* J. Wiley (1991).

[36] Zhang, N., A. Burns and M. Nicholson, "*Pipelined Processors and Worst Case Execution Times,*" Real-Time Systems 5 (Oct. 1993) pp.319-343.