# CONTINUOUS ASSESSMENT OF DESIGNS AND RE-USE IN MODEL-BASED SAFETY ANALYSIS

**Yiannis Papadopoulos[1], Christian Grante[2], Lars Grunske[3], Bernhard Kaiser[4]**

*[1]Department of Computer Science, University of Hull, U.K., y.i.papadopoulos@hull.ac.uk*
*[2]Volvo Cars Corporation, Sweden, cgrante@volvocars.com*
*[3]School of ITEE, The University of Queensland, Brisbane, Australia, grunske@itee.uq.edu.au*
*[4]Fraunhofer IESE, Kaiserslautern, Germany, Bernhard.kaiser@iese.fraunhofer.de*

Abstract: To deliver complex functionalities in a cost effective manner, distributed manufacturing systems should ideally be based on standard interoperable components and be flexible and easily extensible. At the same time, systems must be demonstrably safe and reliable. In this paper, we argue that to balance these conflicting demands effective safety analysis techniques are required that partly automate and simplify off-line safety assessment. We outline a technique that automates the construction of fault trees and FMEAs and explain how this technique can be repeatedly applied in the course of the design life-cycle on functional and architectural models to enable continuous assessment of evolving designs. Finally, we discuss the issue of re-use of safety analyses and give examples of how such reuse simplifies the assessment.

Keywords: safety critical systems, fault identification, fault-tolerance, automated safety analysis, reliability analysis.

## 1. INTRODUCTION AND BACKGROUND

In many engineering and manufacturing industries, the introduction of distributed embedded systems presents opportunities for building cost-effective and flexible design solutions. However, the introduction of such systems in safety critical applications must be carefully considered. In such systems, the integration of functions and its implications, i.e. interaction, interoperation and sharing of resources, raise serious safety concerns which include the possibility of common cause failure and unpredicted dependent failure of critical functions caused by malfunction of non-critical functions. Difficulties are also caused by increasing scale and complexity. The distribution of functions in such systems may indeed be very complex while dynamic reconfigurations may be necessary to meet demands for process changes in real-time. Complete and rigorous safety analyses must, therefore, cover *every* reconfiguration even when this represents only minor changes. Similarly, new safety analyses must be performed every time the design of the system evolves, e.g. to incorporate new requirements, in order to capture correctly any new and potentially hazardous dependencies between functions. Two related questions may be asked at this point concerning our ability to perform safety analysis on complex distributed systems: Can complete and rigorous safety analysis of such systems be achieved within the constraints of modern production? And, if it can, to what extend could safety analyses produced at one stage of the design be re-used to enable the analysis of subsequent versions of an evolving design?

In this paper, we argue that the key to addressing successfully the first question is introducing some degree of automation in safety analysis. In section 2, we illustrate how this can be achieved in the context of HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies) a new technique for semi-automatic construction of fault trees and FMEAs (Papadopoulos, *et al.*, 2001). In sections 3 and 4, we show how application of this technique can be effectively iterated to enable management of design changes in safety analysis. In section 3 we discuss application of HiP-HOPS to abstract functional models, while in section 4 we discuss application to more detailed architectural models. In the respective sections, we also discuss the issue of re-use. As a result of this discussion, we identify two types of reuse which together with automation in the context of HiP-HOPS simplify the analysis of complex systems: re-use of local safety analyses for functions or components that have well-defined operational profiles, and reuse of patterns that describe the failure behaviour of complex components (or fault tolerant schemes) designed to exhibit certain failure behaviour. We also identify the limitations of such re-use and, finally, conclude by pointing out directions of further work.

## 2. HiP-HOPS

HiP-HOPS is a model-based semi-automatic safety and reliability analysis technique. In HiP-HOPS, a structural model of the system (hierarchical if required to manage complexity) is first annotated with formalised logical descriptions of component failures and then used as a basis for the automatic construction of fault trees and FMEAs for the system. Application of the technique can start once a concept of the system under design has been interpreted into an engineering model which identifies components and material, energy or data transactions among components. Suitable models for the application of the technique include functional block diagrams, abstract engineering schematics, piping and instrumentation diagrams, hardware descriptions, data flow diagrams, and other models commonly used in engineering design.

HiP-HOPS can be performed on abstract or more detailed models of the system as these are produced and refined in the course of the design life-cycle. This of course creates opportunities for re-use of earlier analysis and the ability to achieve a consistent and continuous assessment in the centre of which lies the design of the system itself. At the early stages of design, the model that provides the basis for the analysis can be a block diagram which shows the functional composition of the system, input/output transactions among functions and the recursive refinement of functions into networks of lower level sub-functions. Later on, when functions are allocated to hardware, the model becomes a representation of the physical architecture of the system which shows components such as sensors, actuators, busses and programmable controllers enclosing networks of tasks running upon those controllers.

The first step in the analysis of such models in HiP-HOPS is the establishment of the local failure behaviour of each component (i.e. function, hardware or software element) in the model as a set of logical failure expressions which show how output failures of the component can be caused by internal malfunctions and deviations of the component inputs. A variant of Hazard and Operability Studies (HAZOP) is used to identify plausible *output failures* such as the *omission*, *commission*, *value* (hi, low) or *timing* (early, late) failure of each output and then to determine the local causes of such events as combinations of *internal component malfunctions* and similar types of *input failures*. Once this analysis has been completed for all components, and the derived failure expressions have been inserted into the model, the structure of the model is then used to automatically determine how the local failures specified in those expressions propagate through connections in the model and cause functional failures at the outputs of the system. This global view of failure is captured in a set of fault trees which are automatically constructed by traversing the model and by evaluating the local failure expressions encountered during the traversal.

The synthesised fault trees are interconnected and form a directed acyclic graph sharing branches and basic events that arise from dependencies in the model, e.g. common inputs which may cause simultaneous dependent failure of hypothetically “independent” channels in the model. Classical Boolean reduction techniques and recent algorithms for fault tree analysis that employ Binary Decision Diagrams (BDDs) are applicable on this graph. Thus, qualitative analysis (e.g. of abstract functional models) or quantitative analysis (e.g. calculation of system-level failure rates from known failure rates on component-level) can be automatically performed on the graph to establish whether the system meets its safety or reliability requirements. In recent work we have shown that the logic contained in the graph can be automatically translated into a simple table which is equivalent to a classical multiple failure mode system FMEA (Papadopoulos, et al., 2004).

## 3. ITERATION OF HiP-HOPS ON ABSTRACT FUNCTIONAL MODELS AND RE-USE

In HiP-HOPS, human interpretation of the synthesised fault trees and FMEA helps to initiate useful design iterations. To gain maximum value, application of the technique should start as early as possible when abstract functional models of the system become available. The objectives of abstract analysis are two-fold: (a) assist the early identification of design flaws; (b) point out critical functions and guide the design of such functions. Satisfaction of these two objectives would mean that expensive design iterations needed to correct errors late in the design could be avoided while an effective design approach could be established in which the design of critical functions (and safety measures for these functions) could be driven by the result of semi-automatic HiP-HOPS analyses performed on increasingly refined models of the system.

Abstract functional models typically developed at early design stages do not refer to particular hardware architectures, distribution of functions or communication of information. They do identify, though, input, processing and actuator functions, and show how interaction among these functions results in the provision of system functions. In an advanced steer-by-wire system currently designed by Volvo with the aid of the HiP-HOPS tool (see section 5), for instance, such functions include the control of steering, the generation of feedback torque on the steering wheel, and the reception and transmission of information from and to controls in the steering wheel and other locations.

In HiP-HOPS, functional models are first annotated with information about the potential failure behaviour of functions. In the absence of references to particular hardware, analysts can assume that each function potentially exhibits all classes of output failures typically examined in the course of the analysis (i.e. omission, commission, value and timing failures). Failure expressions are then constructed to relate each such output failure to a respective failure mode of the function and deviations of function inputs that can be caused by output failures of other functions further upstream in the model. For a function *X* that receives two inputs *I1* and *I2* and generates an output *O*, an expression for example may define that an omission of *O* (*Omission-O*) can be caused by an omission failure of the function (*OmissionX*) or a simultaneous omission of both inputs (*Omission-I1, Omission-I2*): i.e.:

*Omission-O=OmissionX or Omission-I1 and Omission-I2*

From such expressions describing the potential local failure behaviour of functions, in HiP-HOPS it is possible to automatically generate a set of system fault trees and an FMEA which show how potential failures of input, processing and actuator functions cause system level effects, e.g. in a steer-by-wire system the omission or deviation of steering, driver feedback and other critical system functions. A classification of the severity of those *effects* into marginal and catastrophic can then help to identify the criticality of *causes*, i.e. the criticality of failures of input, processing and actuator functions. These results in turn can guide the design of these functions. Our experience from several rounds of analysis and interpretation of results in the steer-by-wire case study have suggested two general rules that can be applied to directly interpret results from this type of analysis into useful design guidelines:

a) When the analysis indicates that the omission of a function has only marginal effects while commission and value failures have catastrophic effects, a design recommendation should be made to design the function in a way that it “fails silent”. This in turn can lead designers to the identification of new degraded modes in which non-critical functions may fail silent with only marginal effects on the system.

b) When the analysis indicates that all potential failure modes of a function have catastrophic effects on the system then a design recommendation should be made to allocate the function to a fault tolerant architecture.

Application of these guidelines can give new insight into the design of the system. A new system state-chart for example can be constructed to show how graceful transition to the identified degraded modes could be achieved and, driven by these results, design iteration can take place to incorporate these new degraded modes in an improved version of the system model. Further HiP-HOPS studies can be performed on subsequent refinements of the functional model and these can take place before the decision to proceed with the architectural allocation of functions to hardware. In the context of such iteration, the failure annotations of functions that have not been influenced by design changes can be directly re-used. If a function has changed, however, i.e. if it receives different inputs, performs a different transformation on inputs or produces different outputs, then the failure annotations of the function must be revised. To illustrate this, let us take as an example a function *X* that operates on input *I* and generates an output *O*. The following two annotations describe the local failure behaviour of the function:

*Omission-O=OmissionX or Omission-I*
*Commission-O= CommissionX or Commission-I*

Let us now also assume that application of HiP-HOPS has shown that *CommissionX* has catastrophic effects on the system while *OmissionX* has only benign effects and can be tolerated. In response to these results, the design of *X* is revised to include a self-monitoring mechanism at the output. This mechanism detects a condition that would otherwise have been a *Commission-O* and, in return, forces *X* to fail silent. In this new design, a *CommissionX* and the consequent *Commission-O* become implausible and can therefore be removed from the analysis as a potential cause and local effect of failure of *X* respectively. The failure behaviour of the modified function is now described by the following single expression:

*Omission-O=OmissionX or Omission-I or Commission-I*

Such revisions of failure annotations must take place to ensure that the analysis reflects changes in the functional design. However, such changes would typically be limited to areas of the design that the analysis has pointed out to be critical. Hence, substantial re-use of failure annotations, and therefore simplification of the analysis, can be expected in every iteration of the analysis process.

## 4. ITERATION OF HiP-HOPS ON DETAILED ARCHITECTURAL MODELS AND RE-USE

At some point in the development of a design, functions are allocated to hardware components. Input functions are allocated to sensors, output functions to actuators and processing functions are distributed on control processors which are typically connected over one or more busses. Assuming that information about the failure modes of these components is available, HiP-HOPS studies at this stage can become much more detailed and quantitative in nature making use of available information about component failure modes and failure rates.

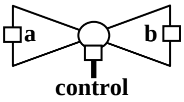
The failure annotations of components are now extended to include the failure modes of each component and any failure rates if available. Such failure modes typically include electrical and mechanical failures caused by wear or environmental conditions for which the component is not qualified. The failure expressions that link component output failures to logical combinations of input failures and internal component malfunctions now make reference to these specific failure modes.

In hierarchical models that record the decomposition of systems, failure annotations can also be inserted at subsystem level to collectively capture the effect of failure conditions that do not necessarily require examination at basic component level. If, for example, a subsystem as a whole is susceptible to some environmental disturbance like electromagnetic interference, then the effects 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 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 sub-system, the fault tree synthesis algorithm creates a disjunction between any failure logic specified at sub-system level and logic arising from the enclosed lower levels. Thus, by enabling causes of failure to be described at both component and sub-system level, it becomes possible to avoid repetition of data that would otherwise be required to describe factors affecting entire sub-systems. This feature, we feel, makes HiP-HOPS a truly hierarchical approach to the analysis of complex systems.

Component failure rates, if provided at this stage, are embedded in the structure of the automatically synthesised fault trees and can be used to perform probabilistic calculations aimed at prediction of the reliability of the system. Note, though, that credible failure rates (often not available) are not essential to produce useful results. Qualitative application of the technique can still produce useful results. The logical reduction of the synthesised fault trees into minimal cut-sets and FMEA, for instance, can indicate single points of failure in the system and point out potential design weaknesses. This, however, should not conclude the analysis process. Indeed, design changes prompted by interpretation of the current analyses may correct existing problems but can also introduce new unintended errors. Thus, re-establishment of the failure behaviour of the system via iteration of HiP-HOPS should follow such design changes. Clearly, our ability to iterate fast this process will ultimately also define our ability to manage effectively the evolution of the design in safety analysis. The automated algorithms of HiP-HOPS for the synthesis of fault trees and FMEAs can clearly help in this direction. However, our ability to effectively apply the method on an evolving design is also heavily dependent upon the ability to re-use the failure annotations that are required by HiP-HOPS at component level.

### *4.1 Reuse of component safety analyses*

It can be easily established that the failure annotations of a component can be directly re-used in the same application as long as design changes do not influence the function of the component, i.e. the component receives the same inputs and performs the same operation. It is also useful to ask at this point whether such annotations could be re-used across different applications. This type of reuse would be more powerful and would clearly help to further simplify safety assessment in the context of HiP-HOPS. To address this question let us examine for a moment the failure annotations of the two-way computer controlled valve illustrated in Figure 1. The figure shows the valve as it would typically be illustrated in a plant diagram and records the results of analysis for the component in two tables that define valve malfunctions and output deviations respectively. In normal operation, the valve is normally closed and opens only when the computer control signal has a value of a logical one (1). Valve malfunctions include mechanical failures such as the valve being *stuckOpen* or *stuckClosed*, and blockages caused by debris such as *blocked* and *partially-Blocked*. For each malfunction, the analysis records an estimated failure rate while the effects of those malfunctions on the output of the valve can be seen in a second table that lists output deviations.



**Valve Malfunctions**

| *Failure mode* | *Description* | *Failure rate* |
|---|---|---|
| Blocked | e.g. by debris | 1e-6 |
| partiallyBlocked | e.g. by debris | 5e-5 |
| stuckClosed | Mechanically stuck | 1.5e-6 |
| stuckOpen | Mechanically stuck | 1.5e-5 |

**Deviations of Flow at Valve Output**

| *Output Deviation* | *Description* | *Causes* |
|---|---|---|
| Omission-b | Omission of flow | Blocked *or* stuckClosed *or* Omission-a *or* Low-control |
| Commission-b | Commission of flow | stuckOpen *or* Commission-a *or* Hi-control |
| Low-b | Low flow | partiallyBlocked *or* Low-a |
| Hi-b | Hi flow | Hi-a |
| Early-b | Early flow | Early-a *or* Early-control |
| Late-b | Late flow | Late-a *or* Late-control |

Fig.1. Failure annotations of a computer-operated two-way valve

Here, we can see that an omission of the output flow (*Omission-b*) can be caused by a number of malfunctions such as valve *blocked* and *stuckClosed* or by input failures such as omission of input flow (*Omission-a*) or a value failure in the control signal (*Low-control*). Similarly, commission of the output flow (*Commission-b*) can be caused by a valve malfunction (*stuckOpen*), commission of input flow (*Commission-a*) or a value failure in the control signal (*Hi-control*). The table also provides expressions that define the causes of value and timing failures at the output of the valve.

In this analysis, there is an implicit assumption that point *b* is always the output of the valve which may be true in a particular system configuration but not in general. To account for flows in the opposite direction, we also need to consider point *a* as an output, which in practice means that the table has to be extended to include deviations of point *a*. The symmetry in the design of the valve means that mechanical replication is only required to complete this specification of how the valve behaves in conditions of failure. This specification 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 component malfunctions and input/output ports of the component. The failure behaviour described in these expressions has been derived assuming a simple operation 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 Fig.1 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. We must stress, thought, that this type of reuse is only possible because the valve has a small number of well-defined interface points and performs the same simple operation in any context of use.

Generalising this discussion, we can say that reuse of failure annotations is likely to be possible for simple components like sensors and actuators. On the other hand, the failure annotations for programmable components or components with variable functional profiles will generally have to be re-constructed each time the component is used in a different application with reference to the functions performed in the context of this application. There are, however, exceptions to this rule and opportunities for reuse of safety analyses even for complex components.

## *4.2 Reuse of patterns of failure behaviour*

Complex components (or sets of such components in fault tolerant configurations) are often designed to provide the same standard failure behaviour on all outputs independently from context of application. A good example of this is the TTA communication controller, a component that handles the communication between a host and other controllers in a time-triggered network (Kopetz, 1995). Analysis of the published specification of this component shows that the controller has several important safety properties that should hold in any context of operation. This, however, enables us to specify a generic (and reusable) pattern of the controller behaviour in the failure domain. The pattern is schematically illustrated in Fig. 2, and shows that the controller is extremely efficient in terms of handling the various classes of failure that analysts would typically examine in a HiP-HOPS study. *Early*, *late* and *commission* failures caused by internal controller faults or the local host are detected and therefore cannot cross the controller-bus interface. Such failures are, therefore, obsolete within the sphere of a TTA network. One reason is that the controller contains mechanisms that prevent the generation of such types of failures. In addition, the controller detects *commission* and *timing* failures generated by the host and transforms these failures into *omissions*. Thus, the only failures that cross the bus interface and enter the sphere of a TTA network are *omission* and *value failures* generated by the host or at the CNI (memory area where host and controller exchange information). Two more types of failure can be generated within the sphere of communications: *omission* and *value* failures caused by external disturbances during transmission (e.g. Electro-Magnetic Interference). From those four classes of failure only one can propagate through and exit undetectable from the TTA network: *value* failures generated by the host or at the CNI.

The pattern of Fig.2 defines a convenient, generic abstraction of the controller behaviour in the failure domain which can be directly translated into failure expressions and then used (and reused) in the context

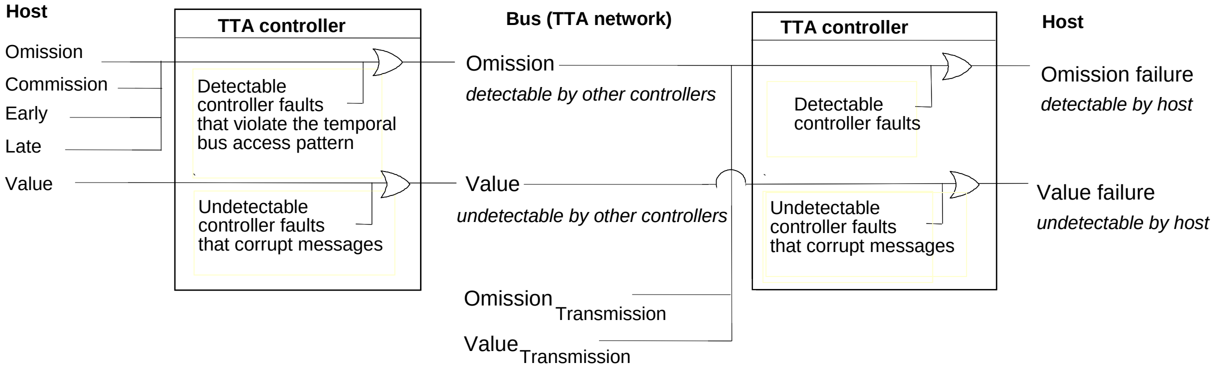

Host
TTA controller
Bus (TTA network)
TTA controller
Host
Omission
Commission
Early
Late
Value
Detectable controller faults that violate the temporal bus access pattern
Undetectable controller faults that corrupt messages
Omission
detectable by other controllers
Value
undetectable by other controllers
Omission_Transmission
Value_Transmission
Detectable controller faults
Undetectable controller faults that corrupt messages
Omission failure
detectable by host
Value failure
undetectable by host

Fig.2. Pattern of failure behaviour of TTA controller (transformation and propagation of failures)

of HiP-HOPS to simplify the safety assessment of TTA networks. However, the assumptions about the properties of the communication controller that underlie this model (stated as requirements in its published specification) must be verified on the actual implementation of the controller before the model can be safely used for all the practical purposes of application safety analysis.

We currently perform a similar type of "pattern" analysis on a number of fault tolerant architectures including schemes that employ hot or cold standbys, majority voters, safety monitors and other mechanisms that enforce certain patterns of failure behaviour (Grunske, 2003). The aim is to develop libraries of such patterns that can be reused in the context of HiP-HOPS in order to simplify the assessment of complex systems.

## 5. TOOL AND APPLICATIONS

To support the proposed process, we have developed a tool that generates system fault trees and FMEAs from Matlab-Simulink models (Papadopoulos and Maruhn, 2001). The synthesised fault trees and their analyses (cut-sets and FMEA) are presented in interactive graphical and tabular form in an HTML viewer. The tool is experimental but usable by third parties and has so far been used in complex case studies (e.g. those reported in Papadopoulos, *et al.*, 2001 & 2004). Volvo is currently evaluating the tool in a study performed on an advanced steer-by-wire prototype. Functional and architectural models of this system have been analysed using this tool. These models are of moderate complexity: they contain hundreds of components and the analysis results to failure logic in the order of tens of thousands of cut-sets. Although initial annotation of such models has proven far from trivial, benefits from reuse meant that substantial reductions in effort were achieved in addition to the benefits of keeping the analyses consistent with the design. Iterations of analyses are still ongoing, but we expect to be in a position soon to publish more on the results of this study, which we hope will shed more light into the application, capabilities and limitations of HiP-HOPS.

## 6. CONCLUSIONS AND FURTHER RESEARCH

In this paper, we proposed a continuous safety assessment process that can guide complex safety critical designs as these naturally evolve in the course of the design life-cycle. The process is based on repeated and consistent application of HiP-HOPS, a new model-based semi-automatic safety and reliability analysis technique that can be applied to progressively more refined models of a system. We have shown that HiP-HOPS is applicable both to abstract functional models and more detailed architectural models. Application to functional models produced early in the design can help to distinguish critical from non-critical functions and drive the design of these functions. On the other hand, application at later stages helps to confirm that design problems have been dealt with effectively and that the system meets its safety and reliability requirements. By automating the synthesis of fault trees and FMEAs, HiP-HOPS are expected to simplify the safety and reliability analysis of complex systems. However manual annotations are required at component level before a synthesis of analyses can be achieved at system level. Re-use of these annotations is therefore crucial in achieving the central objectives of automation and simplification of the analysis. For this reason, we also discussed the potential for, and limitations of, reuse in HiP-HOPS. We identified two types of possible re-use of local analyses: direct re-use of annotations for simple components and re-use of patterns of failure behaviour for more complex components and well-established fault tolerant schemes. We view this second type of reuse as particularly important for the further development of HiP-HOPS as it opens up opportunities for further useful work in the area of automated safety analysis and possibilities for further simplification of safety assessment in the context of this new technique.

We also currently explore the development of automated interfaces between HiP-HOPS and Component Fault Trees a recently proposed extension to fault tree notation that enables modularisation, reuse and fast analysis of failure logic (Kaiser, *et al.*, 2003). In this notation, as in HiP-HOPS, failure logic is developed and structured around the hierarchical decomposition of the system. The two approaches seem to converge and, thus, a potential synthesis and development of interfaces between the respective tools may benefit the overall process of construction and analysis of failure models for complex systems.

## REFERENCES

Grunske, L. (2003). Transformational Patterns for Improvement of Safety Properties in Architectural Specification, 2nd Nordic Conf. on Pattern Languages, Sept. 19-21, Norway.

Kaiser, B., P. Liggesmeyer and O. Mäckel (2003). A New Component Concept for Fault Trees. SCS'03, Canberra, Conf. in Research and Practice in Inf. Tech., **33**, Australian Computer Society.

Kopetz, H. (1995), The Time-triggered Approach to Real-time System Design, *Predictably Dependable Computing Systems*, ESPRIT basic research series, Springer-Verlag, Berlin.

Papadopoulos, Y. and M. Maruhn (2001). Model-based Automated Synthesis of Fault Trees from Simulink models, Int'l Conf. on Dependable Systems and Networks, pp. 77-82, Götenborg.

Papadopoulos Y., J.A. McDermid, R. Sasse and G. Heiner (2001). Analysis and Synthesis of the Behaviour of Complex Systems in Conditions of Failure, RESS, **71**(3):229-247.

Papadopoulos Y., D. Parker and C. Grante (2004). A Method and Tool Support for Model-based Semi-automated FMEA, SCS'04, Brisbane, **38**, Australian Computer Society.

Sinnamon, R.M. and J.D. Andrews (1997). New Approaches to Evaluating Fault Trees, Reliability Engineering and System Safety, **58**:89-96.