Integrating Dependability Analysis into the Design of Distributed Systems[1]

Heinz Kantz
Institut für Technische Informatik
TU - Wien
Vienna, Austria

Institut für Technische Informatik
Technische Universität Wien
Austria

kantz@vmars.tuwien.ac.at

Research Report Nr. 5/91

Mar. 1991

Accepted for publication at the
5th European Computer Conference (COMPEURO91)
May 1991, Bologna, Italy

Institut für Technische Informatik
Technische Universität Wien
Treitlstraße 3/182/1
A-1040 Wien
Austria

Phone: (43) 222-58801-8181
Fax: (43) 222-569149

[1]This work has been supported in part by the ESPRIT Basic Research Project 3092 "Predictably Dependable Computing Systems (PDCS)"

Abstract

The increasing importance and criticality of computers in technical applications have spurred a demand on dependable computers. In these applications acceptance of the system is strongly influenced by dependability aspects. Consideration of dependability has to be an integral part of system development and has to start at early phases in the design.

This paper presents our approach for the integration of dependability analysis into the design of distributed systems. This design methodology supports a close and efficient interaction of design creation and design evaluation activities. The feasibility of this approach is shown by applying it to the application "Rolling Ball", thereby demonstrating the interaction of design creation activities and dependability analysis.

1 Introduction

The tremendous advances in hardware technology and the significant decrease of computers' costs have caused an increased demand on distributed computer systems. Modern technology now allows the use of computer systems to perform complex control task which are no longer manageable by human operators. Moving control from humans to computers increases the computers' importance and criticality. This development forces the designer to deal with dependability [Lap89] aspects and to analyse possible faults and their impact on the system behaviour.

Dependability analysis deals with the estimation and analysis of failures and performs a quantitative assessment of the system. A systematic design of systems is normally characterised by close interaction and integration of design evaluation activities with design creation activities. Dependability analysis applied in early design phases gives only coarse results. Nevertheless, it provides a basis for an early critical examination of the design and results improve as more detailed design information is taken into account. The designer gets a feedback after each design creation step and immediately realises if the current design is unreliable. Furthermore, he gets information for redesigning or optimising the system before it has actually been built. Thus, he avoids costly revisions of major parts of the system.

Dependability analysis has to take into account information about the physical system structure, the software, the description of the fault/error occurrences, the fault/error handling, the repair and maintenance strategy, and the fault hypothesis. This information has to be gathered by the system creation tools and delivered to the dependability analysis tool.

This paper proposes a strategy for integrating dependability analysis into the design of distributed systems based on the MARS design methodology [Sen88]. This methodology has been developed to support the design of distributed hard real-time applications. The distributed real-time architecture MARS (**MA**intainable **R**eal-time **S**ystem) [Kop89] serves as a reference architecture for demonstrating this approach.

The paper is organised as follows: The next section introduces the basic concepts of the MARS architecture. Section 3 starts with a description of the MARS design system (MARDS) and its design steps, then gives a description of the dependability analysis approach in MARS. Section 4 shows the feasibility of this approach in the light of an application, called *Rolling Ball*. The interaction between design creation activities and dependability evaluation activities is

highlighted. Finally, the main points of this paper are summarised.

2 An Overview of MARS

This section provides a brief survey about the main concepts and characteristics of the MARS architecture as necessary for further understanding. For more details the interested reader is referred to [Kop89].

A MARS system is a fault-tolerant distributed real-time system architecture. It is designed to serve as a basic system architecture for real-time applications, where hard deadlines are determined by the environment. Main emphasis is put on a deterministic temporal behaviour, even under anticipated peak load situations.

MARS consists of a set of *clusters*. They may be arbitrarily interconnected via *cluster paths*. Each cluster, consisting of a set of autonomous self-contained computers, called *components*, is a subnet of the MARS system with a high functional connectivity. Components form hardware/software units and are connected within a cluster via a LAN. On each component runs an identical version of the MARS operating system and the component specific application software. Communication among tasks and components is realised solely by broadcasting of messages, via a TDMA (time division multiple access) protocol.

Components are assumed to be *fail-silent*. They either operate as intended or do not produce any results. All internal failures of one component are mapped to one external failure event, the omission of a message which is detected by a membership protocol. In order to achieve a fail-silent property, extensive error detection mechanisms in hardware and software are performed [Kop90a].

Fault-tolerance in MARS is provided with actively redundant components. Due to the fail-silent property of the MARS components, the required service can be delivered as long as at least one of the redundant partner components operates correctly.

3 Dependability Analysis in MARS

3.1 The MARS Design Process

To support the design of MARS applications a graphical Design System, called MARDS [Sen88], has been developed. It assists the entire system development life-cycle covering programming-in-the-large and programming-in-the-small steps by a set of coherent tools for design creation activities and design evaluation activities; such as testing, timing analysis, and dependability analysis. Each of these tools maintains its own local information base and communicates with other tools exclusively via contracts [Dow87]. However, in this paper we are concentrating on the dependability analysis tools and their interaction with the corresponding design creation tools. Other design evaluation activities and their interactions are described in detail in [Kop91].

To describe the activities of a MARS application a transaction oriented model is used. A real-time transaction [Kop90b] is a set of correlated actions between a stimulus and a response. It describes the functional behaviour as well as its timing behaviour. A real-time transaction can be decomposed into a sequence of real-time subtransactions consisting of tasks executions and communication phases between subtransactions. For the rest of this paper we will use the term transaction in the sense of real-time transaction.

One key concept of the design methodology is to reduce the abstraction and complexity by decomposition. For that purpose the whole system and its activities are refined into less abstract subsystems which can be worked on nearly independently. Designing a MARS application is carried out gradually, starting with the requirements definition and ending up with the task implementation. One can easily see that the described methodology chain of designing a MARS application does not consist of rigorously sequential activities. It is likely that some iteration steps are necessary to pass all required quality aspects. It is therefore also possible that some parts of the system are already designed in more detail than other ones, and all tools must be capable of dealing with such incomplete designs.

Each of the design creation activities listed on the left hand side in Figure 1 interacts with the corresponding design evaluation activities. In the following the design creation steps and their interaction with the dependability analysis are described.

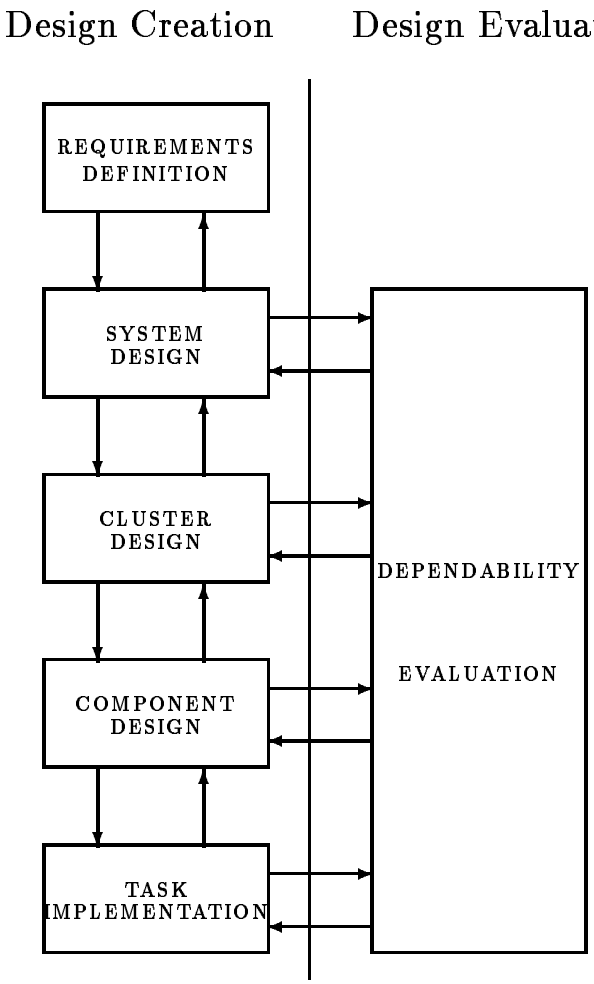

Figure 1: Design Steps

3.1.1 Requirements Definition

Systematic system development starts with requirement engineering. In this phase the designer collects information to understand and formulate the problem domain. The stated problem is described in its functional and timing behaviour via system transactions, required physical elements along with the supplied or consumed data (called data spots), general requirements (e.g. definitions, chosen assumptions, or fault hypothesis), and dependability requirements (e.g. mean time to failure, mean time to catastrophic failure, or percentage of system-up-time). This phase deals with collecting the necessary information and is not directly concerned with design decisions, therefore a dependability analysis is not required. However, specifying and formulating the demanded

dependability properties is a necessary prerequisite for all subsequent steps of a dependability analysis.

3.1.2 System Design and Cluster Definition

In the System Design and Cluster Definition phase the system is decomposed into clusters, and system transactions are refined to cluster transactions. In parallel the dependability requirements are decomposed. The designer can attach estimated dependability attributes to each cluster and to each cluster transaction. At this stage, dependability analysis can be used the first time in order to check if the specified system dependability requirements are still fulfilled under the assumption that each cluster meets the specified dependability requirements. The designer can then use this information as a confirmation or as a starting point for a (partial) redesign of the application.

It is evident that at this high abstraction level dependability analysis serves only as a first step towards a precise and meaningful evaluation. Nevertheless, it forces the designer to tackle this problem domain, and it provides him a first impression of the expected dependability behaviour of the system. Dependability data for clusters and transactions can be estimated from related projects, or derived from standards, e.g. [MHE86].

3.1.3 Cluster Design and Component Definition

The third design creation phase concentrates on the refinement of clusters into components. Concurrently, the cluster transactions are refined to component transactions and are allocated to components.

Now the arrangement and the number of components in the clusters is determined. At this level, components are treated as "fault-tolerant units" (FTUs). Such a FTU is a logical unit which has to fulfill its specified service. The physical realisation (i.e. used hardware, degree and kind of redundancy) is not considered. The FTUs are described by their name, functional attributes, timing attributes, and dependability attributes such as failure behaviour, fault/error handling, impact of a failure on other components and the system, and the repair/maintenance strategy. All this information is refined from the attributes specified in the System Design and Cluster Definition step and is checked for consistency.

Since the system structure on the component (FTU) level is now fixed, a more detailed dependability analysis can be initiated. Different architectural

variants, parameter variations, and sensitivity analysis reveal the critical parts of the system and suggest ways for their improvement.

3.1.4 Component Design and Task Definition

In the Component Design and Task Definition phase the inner structure of the FTUs is detailed. The designer does now specify the physical realisation and the degree and kind of redundancy. In MARS only actively redundant components are used. Active redundancy avoids in the presence of faults a reboot or synchronisation of the redundant component, thus guaranteeing a continuous service. However, the design methodology itself does not preclude the use of passive redundancy or other redundancy mechanisms as hot-standby components or voting.

The designer takes into account precise dependability parameters of the hardware as well as estimated dependability parameters of the defined tasks. After an analysis of this configuration it can now be checked if the hardware fulfills the dependability requirements as derived from the previous design phase. If this is not the case a reconfiguration of the FTUs, by adding additional redundancy or using more reliable components, can be performed. If even reconfiguration of the FTUs does not succeed, a redesign on a higher level (e.g. Cluster Design and Component Definition) is initiated. It is noteworthy to mention that at this stage no building of hardware or coding activities have been initiated so far. A redesign is still relatively cost-effective.

3.1.5 Task Implementation

The final step of the methodology chain deals with the implementation of the single tasks as derived from the Component Design and Task Definition level. At this stage, the impact of software on the system is not yet applicable. Therefore as a first approach software is treated similar as hardware. After extensive test runs during the Test phase [Sch90], the software failure behaviour can be quantitatively assessed from the data obtained from the error history.

3.2 The Dependability Analysis Process

Traditionally, dependability analysis is carried out as follows: First, the system is analysed and the relevant system properties are identified. Then this information is used to construct dependability models (e.g. Markov models,

Stochastic Petri Nets, Combinatorial models) which can be analysed with the help of an evaluation tool.

We do not expect a designer to be experienced in dependability modelling. Thus, the designer needs to be supported in the modelling process. Especially, he needs to be relieved in the tedious and error-prone task of dependability model specification and construction. Thus, the input of the dependability evaluation process should rather be a design description than a dependability model which can be specified by a designer with a basic knowledge in dependability modelling. This approach is not entirely new, recent developments of tools for dependability evaluation revealed the need for assisting the input of dependability models and provide high-level input description languages, see [Bab87,Joh88,Gei90]. For example, the tools HARP, METFAC, or SAVE provide a special input language allowing a compact and comfortable model specification. The tools METASAN and SPNP accept extensions of stochastic Petri Nets as input. All these tools do internally transform their high level model description into a Markov model and solve it afterwards.

This consideration results in the development of a two pass dependability evaluation process [Mul88a] closing the gap from the model specification to the model evaluation. The interface between the design creation tools (from System Design to Task Implementation) and the dependability analysis tools is defined as a detailed description of the designed system in the form of a structured, text-oriented, description language. This general design description is augmented with dependability data including failure distributions of objects, coverage values, and degree and kind of redundancy.

In the first pass, this formalised design description is transformed into a dependability model. The second pass evaluates this automatically generated model. Besides the advantage of assisting even a designer who has only a basic knowledge in dependability modelling, this approach supports the concept of modularisation. The input of the dependability analysis is independent of the particular underlying evaluation method. Thus, additional evaluation tools may be easily incorporated, as only the first pass has to be modified.

In our prototype implementation the first pass transforming the design description into a dependability model is carried out by a tool called MARPLE [Mul88b]. The basic elements of the MARPLE input language are objects (such as clusters, components, sensors, and tasks), information items (notions for the abstract entities of information exchange), and transactions. The structure of the language is invariant during all design creation phases, allowing an equal treatment of all basic elements.

MARPLE transforms this design description into PMS (Processor-Memory-Switch) models [Kan89a,Kan89b]. The second pass is carried out by the dependability evaluation tool SHARPE [Sah87], which provides a hierarchical modelling framework of Markov models and Combinatorial models such as PMS-models, Fault-trees, Series-parallel block diagrams, or Series-parallel directed acyclic graphs.

This approach is very powerful, but also has its limitations. First, it is often desirable to analyse a more detailed and more complex system behaviour than it can be expressed with the MARPLE input language (e.g. the modelling of stress factors or near-coincident failures). In order to support this aim the designer can reference predefined dependability models. These models, which may be independently developed or derived from related projects, can be easily included. This approach not only supports the expression of complex system behaviour, it also supports the concept of model reusability. Models from related projects can be stored in an archival database and reused on demand. The second point is that automatic generated models tend to be less efficient in their evaluation than models which have been specified by a modelling expert. For efficiency reasons the designer has the option to include such manually generated models.

4 The Rolling Ball Application

In this section we will demonstrate the feasibility and practicability of integrating dependability analysis into system design. As an example we choose a real-time application [Kop90c] called "Rolling Ball". The aim of this application is to balance a ball on a flat plane such that it follows a cycle with predefined radius and speed. With the help of a video camera the coordinates of the ball are scanned periodically and the required inclination of the plane is computed and output to four servo motors. This implementation serves as a testbed for measurements and experimental evaluations.

In the following we will demonstrate our approach for integrating dependability analysis into the system design of the Rolling Ball application. We will discuss the design process and its interaction with dependability analysis from the System Design and Cluster Definition phase to the Task Implementation. Due to the length limitation the transformation of the system description (MARPLE input) into a PMS-model, the evaluation of this model type in SHARPE, and a discussion of the results are beyond the scope of the pa-

per. The interested reader can find an extensive discussion of these points in [Mul88a,Kan89a,Kan89b].

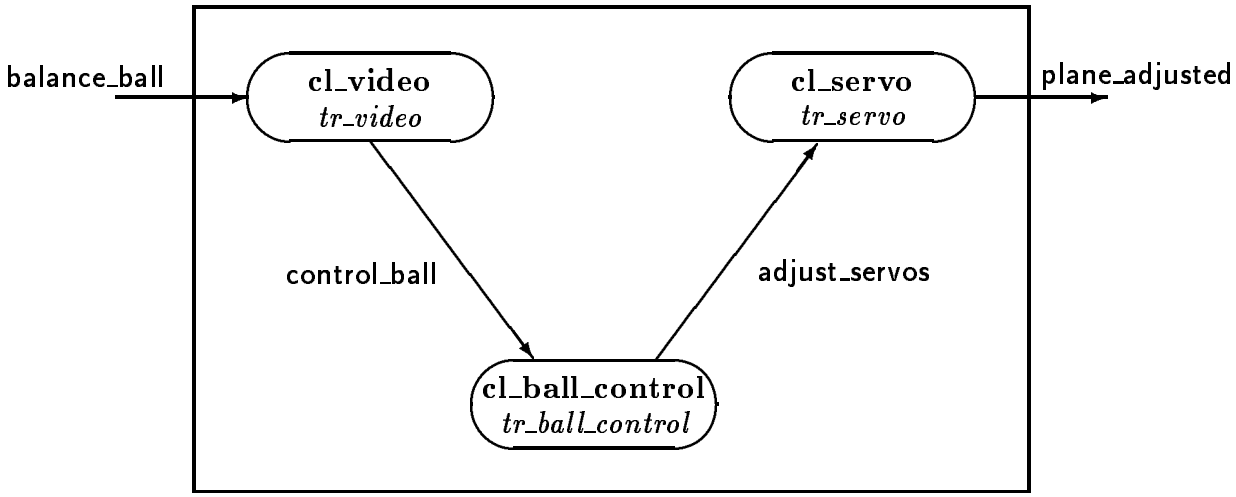

Figure 2: System Rolling Ball

4.1 System Design and Cluster Definition

As a first decomposition step, the system is divided into 3 clusters: **cl_video**, **cl_ball_control**, and **cl_servo**, as shown in Figure 2. The balancing of the ball is described by the system transaction *tr_balance*. This transaction is triggered through the stimulus balance_ball and results in the response plane_adjusted. During the Cluster Definition the transaction *tr_balance* is detailed into three cluster transactions: *tr_video, tr_ball_control,* and *tr_servo*.

The cluster **cl_video** is responsible for the observation and recognition of the rolling ball on the plane. This activity is expressed by the cluster transaction *tr_video* which consumes the information item balance_ball and produces the information item control_ball. The response of this transaction acts as a stimulus for transaction *tr_ball_control* which is located on the cluster **cl_ball_control** and calculates the required inclination of the plane to keep the ball on the specified course. Transaction *tr_ball_control* produces the response adjust_servos which is used as a stimulus for transaction *tr_servo*. This cluster transaction running on cluster **cl_servo** describes the controlling of the servo units.

Each cluster and cluster transaction is associated with estimated dependability attributes, serving as input for the dependability analysis. In this application we are interested in the reliability of the system, thus attaching each cluster with a failure rate. However, at this stage the analysis provides only a first step towards meaningful results. Due to the tight space limit of the paper we do not discuss the MARPLE input at this level.

To keep this example small, in the following we will only discuss the refinement of the **cl_ball_control** cluster in detail. The decomposition of the other two clusters, **cl_video** and **cl_servo** is explained informally and is not shown as MARPLE input.

4.2 Cluster Design and Component Definition

In this phase, the 3 clusters are refined to components and the 3 cluster transactions are refined to component transactions. In parallel, the dependability attributes for the clusters are refined to component level attributes.

The **cl_video** cluster consists of two components: a **video camera** and a **PC**. The video camera is connected to the PC which receives the video signals from the camera. The PC then determines the position of the ball and transmits its coordinates to the **cl_ball_control** cluster.

The cluster **cl_ball_control** is refined into 3 components: **co_video**, **co_calculation**, and **co_control**. The functionality of the whole cluster is expressed by the cluster transaction *tr_ball_control*. This cluster transaction is decomposed into a linear sequence of 3 component transactions: *tr_ball_control.1*, *tr_ball_control.2*, and *tr_ball_control.3*. The corresponding MARPLE input of this cluster is listed in Figure 3.

The **co_video** component receives the pixel coordinates from the **PC** over a serial line and transforms them into real world coordinates. This activity is described as transaction *tr_ball_control.1*. The coordinates form the input of the **co_calculation** component which calculates the acceleration of the ball. This calculation is described as transaction *tr_ball_control.2*. The component **co_control** transforms the acceleration into impulse lengths for the cluster **cl_servo**, (see transaction *tr_ball_control.3*). For proper operation of the cluster the whole sequence of component transactions needs to be operating which is expressed with the condition statement in the transaction *tr_ball_control* definition part in the MARPLE input.

In parallel to the refinement of the components and transactions the de-

pendability attributes are refined. For this application we require a constant failure rate $\lambda = 1/(6 * 8760)\ h^{-1}$ for each of the three components, which represents a MTTF of 6 years.

In this application all components are required for system operation, and assuming a statistically independent failure behaviour of the components, they can be seen as a series block diagram. In this example, the transactions are refined to a sequence of subtransactions. However, the methodology is not restricted to a serial refinement only.

```
object cluster cl_ball_control
    consumes rolling_ball/control_ball
    produces rolling_ball/adjust_servos

    transaction tr_ball_control
        consumes rolling_ball/control_ball
        produces rolling_ball/adjust_servos
        condition ./co_video/tr_ball_control.1 and ./co_calculation/tr_ball_control.2
                  and ./co_control/tr_ball_control.3

    object component co_video
        consumes rolling_ball/control_ball
        produces  ../s1

        transaction tr_ball_control.1
            consumes rolling_ball/control_ball
            produces  ../s1

        physobj
            lambda 1/(6*8760)
    end object component

    object component co_calculation
        consumes ../s1
        produces  ../s2

        transaction tr_ball_control.2
            consumes ../s1
            produces  ../s2

        physobj
            lambda 1/(6*8760)
    end object component

    object component co_control
        consumes ../s2
        produces  rolling_ball/adjust_servos

        transaction tr_ball_control.3
            consumes ../s2
            produces  rolling_ball/adjust_servos

        physobj
            lambda 1/(6*8760)
    end object component

end object cluster
```

Figure 3: MARPLE input of cluster cl_ball_control

The **cl_servo** cluster consists of a **controller** and 2 **servo units** for controlling the plane. The controller receives the impulse length from the cluster **cl_ball_control** and adequately controls the servo motors for the corresponding inclination of the plane.

4.3 Component Design and Task Definition

In this phase the internal structure of the components, which has been defined on the logical level in the previous design phase, is now defined. The **video camera** and the **PC** are non-redundant components. The components of the **cl_ball_control** cluster operate in two-fold active redundancy.

The MARPLE input language supports the expression of FTUs consisting of actively or passively redundant components. However, in this application we express the reliability behaviour of the FTUs in the cluster **cl_ball_control** with a predefined submodel. This approach allows the description of complicated reliability behaviour, such as failure dependencies, stress factors, or near coincident failures. To this purpose we have defined Markov graphs for modelling different types of FTUs, as described in [Kop90a]. These models include failure rates for permanent failures and transient failures with corresponding repair rates, as well as coverage values that a single failure is handled correctly. The only difference in the MARPLE input, shown in the Cluster Design and Component Definition phase, is that instead of defining the components as physical objects with the keyword **physobj** and a failure distribution, the predefined dependability model is referenced via the filename and the name of the particular model.

The **servo units** consists of two units for controlling the two degrees of freedom of the plane with two **servo motors** each, the **controller** operates non-redundantly. The **cl_servo** cluster requires the **controller** and 3 out of 4 **servo motors** for proper operation.

4.4 Task Implementation

The last design step deals with the implementation of the component transactions. In the current approach software is treated in the same way as hardware, being attached with dependability attributes.

Before coding activities start the designer has to deal with estimated values. More meaningful values can be obtained in analysing the error history of extensive test runs. Software for this project is currently still under development,

and we have not yet derived significant data for it. Therefore this design phase is not discussed here.

5 Summary

System design of distributed computer systems can no longer be limited to support only pure design creation activities. Moreover, systematic design is characterised by a coherent treatment of design creation activities and design evaluation activities. In order to close the gap between these activities a close coordination and interaction of all design activities in a comprehensive design environment is necessary.

This paper has presented by means of the Rolling Ball application a strategy for integrating a dependability analysis in the design of distributed computer systems. In this application an integration of dependability analysis in all design activities starting from the System and Cluster Definition to the Component Design and Task Definition has been discussed.

Currently, as a first approach, software is treated as hardware. Future work will include an study on software and software fault-tolerance techniques as N-version programming [Avi85], N-Self-Checking Programming [Lap87], or Recovery Blocks [Ran87] and their impact on the system.

The experiences from the Rolling Ball application have shown that such an approach is practically feasible and also appropriate for more complex applications.

6 Acknowledgment

The author wishes to thank W. Schütz, A. Vrchoticky, and R. Zainlinger for their valuable and helpful comments on earlier versions of this paper.

References

[Avi85] A. Avizienis. The N-Version Approach to Fault-Tolerant Systems. *IEEE Transactions on Software Engineering*, 11(12):1491–1501, Dec. 1985.

[Bab87] P. Babcock, F. Leong, and E. Gai. On the next Generation of Reliability Analysis Tools. Technical report, NASA Contractor Report 178380, Oct. 1987.

[Dow87] M. Dowson. ISTAR and the Contractual Approach. *Communications of the ACM*, 30(3):287–288, Mar. 1987.

[Gei90] R. Geist and K. Trivedi. Reliability Estimation of Fault Tolerant Systems: Tools and Techniques. *IEEE Computer*, 23(7):52–61, July 1990.

[Joh88] A. M. Johnson and M. Malek. Survey of Software Tools for Evaluating Reliability, Availability and Serviceability. *ACM Computing Surveys*, 20(4):227–269, Dec. 1988.

[Kan89a] H. Kantz and M. Mulazzani. Modeling of PMS - Structures in SHARPE. In *IFAC Proceedings SAFECOMP 89*, pages 97–102, Vienna, Austria, Dec. 1989.

[Kan89b] H. Kantz and M. Mulazzani. PMS - Models in SHARPE. In *Proc. 4th International Conference on Fault-tolerant Computing Systems*, pages 256–267, Baden-Baden, Germany, Sep. 1989.

[Kop89] H. Kopetz, A. Damm, Ch. Koza, M. Mulazzani, W. Schwabl, Ch. Senft, and R. Zainlinger. Distributed Fault-Tolerant Real-Time Systems: The MARS Approach. *IEEE Micro*, 9(1):25–40, Feb. 1989.

[Kop90a] H. Kopetz, H. Kantz, G. Grünsteidl, P. Puschner, and J. Reisinger. Tolerating Transient Faults in MARS. In *Proc. 20th Int. Symposium on Fault-Tolerant Computing*, pages 466–473, Newcastle upon Tyne, UK, June 1990.

[Kop90b] H. Kopetz and K. Kim. Temporal Uncertainties in Interactions among Real-Time Objects. In *Proc. 9th Symposium on Reliable Distributed Systems*, pages 165–174, Huntsville, AL, USA, Oct. 1990. IEEE Computer Society Press.

[Kop90c] H. Kopetz, R. Noisser, et al. The Rolling Ball: A Real-Time Control Problem. Research Report 2/90, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, 1990.

[Kop91] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, and W. Schütz. An Engineering Approach towards Hard Real-Time System Design. Research Report 1/91, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, Jan. 1991.

[Lap87] J. C. Laprie, J. Arlat, C. Beounes, K. Kanoun, and C. Hourtolle. Hardware- and Software-Fault Tolerance: Definition and Analysis of Architectural Solutions. In *Proc. 17th Int. Symposium on Fault-Tolerant Computing*, pages 116–121, Pittsburgh, PA., July 1987.

[Lap89] J. C. Laprie. Dependability: A Unifying Concept for Reliable Computing and Fault Tolerance. In T. Anderson, Editor, *Dependability of Resilient Computers*, pages 1–28. BSP Professional Books, Oxford, 1989.

[MHE86] MIL-HDBK-217E. *Reliability Prediction of Electronic Equipment.* U.S. Department of Defense, Oct. 1986.

[Mul88a] M. Mulazzani. *Generation of Dependability Models from Design Specifications of Distributed Real-Time Systems.* PhD Thesis, Technisch Naturwissenschaftliche Fakultät, Technische Universität Wien, Vienna, Austria, Apr. 1988.

[Mul88b] M. Mulazzani. An Open Layered Architecture for Dependability Analysis and its Application. In *Proc. 18th Int. Symposium on Fault-Tolerant Computing*, pages 96–101, Tokyo, Japan, June 1988.

[Ran87] B. Randell. Design-Fault Tolerance. In J. C. Laprie A. Avizienis, H. Kopetz, Editor, *The Evolution of Fault-Tolerant Computing*, pages 251–270. Springer Verlag, Vienna, Austria, 1987.

[Sah87] R. Sahner and K. S. Trivedi. Reliability Modeling using SHARPE. *IEEE Transactions on Reliability*, 36(2):186–193, June 1987.

[Sch90] W. Schütz. A Test Strategy for the Distributed Real-Time System MARS. In *IEEE CompEuro 90, Computer Systems and Software Engineering*, pages 20–27, Tel Aviv, Israel, May 1990.

[Sen88] Ch. Senft. A Computer-Aided Design Environment for Distributed Realtime Systems. In *IEEE CompEuro 88, System Design: Concepts, Methods and Tools*, pages 288–297, Brussels, Belgium, Apr. 1988.