Research Reports ESPRIT

Project 818/2252 · Delta-4 · Vol. 1

Edited in cooperation with
the Commission of the European Communities

D. Powell (Ed.)

Delta-4: A Generic Architecture for Dependable Distributed Computing

Springer-Verlag
Berlin Heidelberg New York London Paris
Tokyo Hong Kong Barcelona Budapest

Editor

David Powell
LAAS-CNRS
7, avenue du Colonel Roche
F-31077 Toulouse, France

ESPRIT Project 818/2252 "Definition and Design of an Open Dependable Distributed Computer System Architecture (Delta-4)" belongs to the Subprogramme "Advanced Information Processing" of ESPRIT, the European Strategic Programme for Research and Development in Information Technology supported by the Commission of the European Communities.

The Delta-4 objectives are to formulate, develop and demonstrate an open, fault-tolerant, distributed computing architecture. The key features of the architecture are: a distributed object-oriented application support environment, built-in support for user-transparent fault-tolerance, use of multicast or group communication protocols, and use of standard off-the-shelf processors and standard local area network technology with minimum specialized hardware.

CR Subject Classification (1991): C.2, D.4.5, D.4.7, B.6.2

ISBN 3-540-54985-4 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-54985-4 Springer-Verlag New York Berlin Heidelberg

Publication No. EUR 14025 EN of the
Commission of the European Communities,
Scientific and Technical Communication Unit,
Directorate-General Telecommunications, Information Industries and Innovation,
Luxembourg
Neither the Commission of the European Communities nor any person acting on behalf of the Commission is responsible for the use which might be made of the following information.

Printed in Germany

Typesetting: Camera ready by author
Printing and Binding: Weihert-Druck GmbH, Darmstadt
45/3140 – 543210 – Printed on acid-free paper

FOREWORD

Delta-4 is a 5-nation, 13-partner project that has been investigating the achievement of dependability in open distributed systems, including real-time systems.

This book describes the design and validation of the distributed fault-tolerant architecture developed within this project. The key features of the Delta-4 architecture are: (a) a distributed object-oriented application support environment; (b) built-in support for user-transparent fault-tolerance; (c) use of multicast or group communication protocols; and (d) use of standard off-the-shelf processors and standard local area network technology with minimum specialized hardware.

The book is organized as follows:

The first 3 chapters give an overview of the architecture's objectives and of the architecture itself, and compare the proposed solutions with other approaches.

Chapters 4 to 12 give a more detailed insight into the Delta-4 architectural concepts. Chapters 4 and 5 are devoted to providing a firm set of general concepts and terminology regarding dependable and real-time computing. Chapter 6 is centred on fault-tolerance techniques based on distribution. The description of the architecture itself commences with a description of the Delta-4 application support environment (Deltase) in chapter 7. Two variants of the architecture — the Delta-4 *Open System Architecture* (OSA) and the Delta-4 *Extra Performance Architecture* (XPA) — are described respectively in chapters 8 and 9. Both variants of the architecture have a common underlying basis for dependable multicasting, i.e., an atomic multicast protocol and fail-silent hardware; these are described respectively in chapters 10 and 11. The important topic of input to, and output from, the fault-tolerant Delta-4 architecture is tackled in chapter 12.

Chapters 13 and 14 describe the work that has been carried out in the fields of security, by intrusion-tolerance, and the tolerance of software design faults by diversified design.

Finally, chapter 15 gives an extensive overview of the validation activities carried out on various sub-systems of the architecture.

Several annexes that detail particular points are included together with a glossary of abbreviations that are used throughout the book.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the strong support given to the members of the project by their respective organizations: *Bull SA, Crédit Agricole, Ferranti International plc, IEI-CNR, IITB-Fraunhofer, INESC, LAAS-CNRS, LGI-IMAG, MARI Group, SRD-AEA Technology, Renault Automobile, SEMA Group* and the *University of Newcastle-upon-Tyne*. The enthusiastic and close collaboration between partners, which this backing has encouraged from the outset, was an important factor in achieving the project objectives.

In particular, the authors acknowledge the work and the support of: *Martine Aguera, Marc Alciato, Susanne Angele, John Baldwin, Yada Banomyong, Mario Baptista, Gilles Bellaton, Patrick Béné, David Briggs, Derek Brown, Errol Burke, Gérard Carry, Jean Catala, John Clarke, Alain Costes, Michael Dean, Ken Doyle, John Etherington, Paul Ezhilchelvan, Joseph Faure, Jean-Michel Fray, Jacques Goze, Frédéric Grardel, Georges Grunberg, Christian Guérin, Jerôme Hamel, Rainer Helfinger, Alain Hennebelle, Peter Hinde, Wolfgang Hinderer, Alex Hogarth, Marie-Thérèse Ippolito, Ross Kavanagh, Ivor Kelly, Philippe Lebourg, Marc Liochon, Gilles Lorenzi, Pascal Martin, Navnit Mehta, Denis Michaud, Stephen Mitchell, Luís Nacamura, Christopher Nichols, Gilles Pellerin, Joëlle Penavayre, Eliane Penny, Laurence Plancher, Bernard Quiquemelle, Pierre-Guy Ranéa, Alan Reece, John Reed, Philippe Reynier, René Ribot, David Riley, Bruno Robert, Bethan Roberts, Carlos Rodriguez, Paul Rubenstein, Heike Schwingel-Horner, Bruno Séhabiague, Philippe Sommet, Isabelle Theiller, Touradj Vassigh, David Vaughan, François Waeselynck* and *Jiannong Zhou*.

We also wish to acknowledge the contribution made in previous phases of this project by our earlier partners: *BASF, GMD-First, Jeumont Schneider, Telettra* and the *University of Bologna*.

* * *

The Delta-4 project was partially financed by the *Commission of the European Communities* as projects 818 and 2252 of the European Strategic Programme for Information Technology (ESPRIT).

LAAS-CNRS would like to acknowledge additional sponsorship of the Delta-4 project from the *Conseil Régional de Midi-Pyrénées*.

* * *

TABLE OF CONTENTS

Chapter 15

Validation

Users of the Delta-4 architecture must be able to have a *justified* confidence in its dependability. Consequently, such an architecture must undergo extensive *validation* both from the *verification* and *evaluation* viewpoints.

Verification is that part of the validation activity aimed at removal of design and implementation faults. Verification in Delta-4 is carried out at two levels:

- verification of the design of the communication protocols to discover and remove design faults,
- verification of the implementation by means of injection of hardware faults to verify the effectiveness of the architecture's self-checking and fault-tolerance mechanisms.

Dependability evaluation is that part of the global activity of validation that pertains to fault-forecasting, i.e., the estimation of the presence, the creation and the consequence of faults. Dependability evaluation is also carried out at two levels:

- evaluation of dependability measures of Delta-4 architecture configurations taking into account the nature of the different elements (e.g., fail-silent or fail-uncontrolled hosts, replication domain of the different components, replication techniques, reconfiguration possibilities, repair policies, ...)
- evaluation of software reliability through the application of reliability growth models.

The aim of each validation activity and the main results are summarised in the next section. Then, sections 14.2 - 14.5 provide further details on each activity.

15.1. Overview

Section 15.2 is devoted to protocol verification. The need to assure reliable communication among distributed sites has led to the design of specific protocols for the Delta-4 architecture, such as the Atomic Multicast protocol (AMp, cf. chapter 10) and the Inter Replica protocol (IRp, cf. section 8.1). To improve the quality of these protocols, formal methods have been used for their specification and verification.

The specifications of a protocol consist of the formal description of the distributed algorithm, the formalization of the assumptions about its execution environment and the formal definition of properties characterizing the service it should deliver. Formal specifications are useful on their own, since they force the specifier to formally explicit crucial features of the protocols. Furthermore, the verification of their consistency allows detection of possible errors early in the software development. We are concerned with *formal verification*, that means:

> Given some description of an algorithm and given a description of its "service specifications", verify formally that the described algorithm delivers the specified service.

Verification is carried out by model checking techniques using the *Xesar* verification toolbox. This toolbox was developed to evaluate properties — given by formulas of temporal logic — on a model obtained from a program describing the algorithm to be verified (embedded in its execution environment). These methods have important advantages for discovering design faults since they are based on a complete search of the graph of all possible behaviours of the system to be verified.

The AMp and IRp families of protocols have been specified and verified. Inconsistencies of different nature have been detected such as incorrect initializations of local variables, or too weak conditions. Some of them are only detectable in some peculiar sequence of events that it would be unlikely to obtain by simulation, such as unspecified receptions, non termination of the monitor election phase or duplication of messages. Implementations have been derived from the formal specifications.

The section relative to formal verification overviews the method followed and contains the major results obtained for the critical protocols that have been studied. Annexes K and L complement this section by surveying the various models that can be used to represent reactive systems and techniques for specifying system behaviour.

Section 15.3 is dedicated to implementation validation by means of fault injection. Work on implementation validation is aimed at testing the basic building blocks that support the fault tolerance features of the Delta-4 architecture: the fail-silent assumption of the NAC components and the fault resiliency provided by AMp. The experimental validation carried out is based on the use of physical fault injection, i.e., physical faults are directly injected on the pins of the circuits that constitute the NAC.

The distributed testbed, including the fault injection tool *MESSALINE*, that has been developed to carry out this work provides an experimental environment that enables not only successive versions of the AMp software to be addressed[1], but also two distinct versions of the NAC hardware: one with restricted self-checking mechanisms (referred to as a *fail-uncontrolled* NAC) and another with improved self-checking (referred to as a *fail-silent* NAC).

The section relative to fault-injection validation contains an overview of the method, a brief description of the testbed and the major results obtained so far (essentially, those for the fail-uncontrolled NAC).

Section 15.4 deals with dependability modelling and evaluation. The objective of this activity is to provide the users with a quantified assessment of the amount of dependability that the architecture provides, i.e., the degree by which they can justifiably rely on the architecture.

Dependability modelling is based on Markov processes. Use of Markov process is first justified and general expressions giving the equivalent failure rate and the steady-state or asymptotic unavailability are presented.

Delta-4 is a modular and open architecture and all possible configurations have a common point: a communication support system that constitutes a hard core since its failure leads to loss of interactions between the hosts. In this section, emphasis is put on the selection of the "best" architecture among all possible ones.

Various communication topologies are considered (802.4 token bus, and 802.5 and FDDI token rings), for each of them a single and a double configuration architecture is modelled. Two dependability measures are evaluated: the equivalent failure rate and the asymptotic unavailability.

---

1 Obtained in particular as a result of the corrections induced by the design errors detected during this validation phase.

It is shown that for the single media configurations the equivalent failure rates are limited by the failure rate of the medium and non-covered failures of the NACs and that, for the double media configurations, they are directly related to the failure rate of the non-covered failures only.

Comparison of these architectures showed that it is very difficult to classify them: the evaluated measures depend on several parameters and a tradeoff is needed to select the most suitable architecture. However it is shown that — whatever the architecture — the coverage factor of the NAC is of prime importance, it is thus worthwhile to put emphasis on this coverage (i.e., self-checking mechanisms) during development.

Finally, *section 15.5* is devoted to software reliability evaluation. Quantitative assessment of software reliability is usually carried out through the application of reliability growth models. These models enable prediction of either the number of failures to be activated for the next period of time or the mean time for the next failures or the software failure rate or some combinations of these measures.

Reliability growth models are generally parametric models and these parameters have to be estimated (i.e., model calibration) to carry out dependability predictions. Calibration of the models is fulfilled through failure data collected on the software either during development or in operation. Predictions are thus based on the observation of the software behaviour during a given period of time, calibration of the model using the observed failure data and application of the model to estimate dependability measures.

Data collection is a long process and is now being carried out on the developed software. Reliability growth models will be applied when sufficient data items have been collected.

Since insufficient data has been collected at the time of writing, this section is mainly devoted to data collection and the problems that this data collection induces.

15.2. Protocol Validation

15.2.1. Introduction

In the Delta-4 project, the need to assure reliable communication among distributed sites has led to the definition of specific protocols, such as the Atomic Multicast protocol (AMp) or the Inter Replica protocol (IRp). As the system architecture is based on these protocols, the quality of their development is crucial. One approach for improving quality is to carry out simultaneously the design and its verification by using formal methods and then to derive the implementation.

The usual informal specifications given in an implementation guide are not sufficient to achieve formal verification. The first task is to provide a structured formal description of the protocols and of the delivered services. These formalizations are useful on their own, as they enforce the specifier to explicit the assumptions on which the system is based. Furthermore, it is important to verify formally the consistency of the service definition with the protocol description to detect possible inconsistencies early in the software development activity.

15.2.1.1. Formal Verification Techniques. Formal verification of a design requires a description of an algorithm and a description of its "service specifications". The aim is then to verify formally that the described algorithm delivers the specified service.

The systems we are interested in interact with their *environment* and thus cannot be adequately described by a purely functional view, in terms of input/output. Typical examples are operating systems, communication protocols, distributed data bases, digital systems ... These systems, which are better specified in terms of their behaviours, are known as *reactive systems* [Pnueli 1986]. There is a large agreement on the fact that the algorithms involved in

such systems are highly complex, and their development needs a large effort, especially concerning formalization and verification of their design. Furthermore, the verification of time bounds in the distributed algorithms used in communication protocols is crucial.

Two main classes of techniques have been proposed for formal verification:

- Program verification based on *deductive methods* [Misra and Chandy 1981, Owicki and Gries 1976]: a distributed system is described as an abstract program, and the service to be delivered is characterized by a set of properties, described for example by formulas of temporal logics [Pnueli 1977], which are transformed into assertions about the program. The proof of these assertions can be partially automated by using theorem provers, e.g., [Boyer and Moore 1979, Gordon et al. 1979]. These methods extend to concurrent programs the methods proposed by Floyd [Floyd 1967] and Hoare [Hoare 1972] for sequential programs.
- *Model checking* techniques consisting in: building a model from the description of the distributed system, and checking the service properties on this model by using appropriate algorithms. Usually these methods are restricted to finite state programs and are suitable for protocols. Many proposals based on this approach have been made, for example [Clarke et al. 1986, Fernandez et al. 1985, Holzmann 1984, West 1982, Zafiropulo et al. 1980]. These techniques are more adequate for automatic verification of large systems.

Both approaches require:

- A formal description of the algorithm under study, given, e.g., in the form of a labelled transition system or a set of equations, or a CCS [Milner 1980], CSP [Hoare 1985], Lustre [Caspi et al. 1987], Estelle [ISO 9074] or Lotos [ISO 8807] program. In the case of model checking, this description is translated into a *model* (see annexe K) representing the behaviour of the algorithm.
- Formal service specifications, given as a set of properties characterizing this service, or as an abstract algorithm describing it. In the case of model checking, each property is evaluated on the model.
- The assumptions about the *execution environment* of the algorithm.

For the result of the verification to be meaningful, some *soundness condition* must be satisfied. One possible soundness condition is that all properties that are formally verified must be true in reality. If one is more interested in error detection, the soundness condition may also be formulated the other way round, i.e., each error detected in the verification phase must correspond to a real error.

15.2.1.2. Verification of Timed Algorithms. Usual methods for reasoning about reactive systems abstract away from quantitative time, preserving only ordering properties, such as "whenever a process receives a message, it has been sent previously by another process". However, in the domain of distributed systems, especially fault-tolerant ones, the notion of time is important.

First, two remarks should be made:

- The notions of time in the algorithm and in the model need not to be the same.
- Formal verification is different from performance evaluation. Formal verification is performed on an abstraction of an implementation. Performance evaluation works on a particular implementation, for which intervals of execution times for all basic actions are supposed to be known, and thus an exact picture of the time behaviour may be obtained.

The notion of time needed for the verification depends obviously on the notion of time needed by the described algorithm. A classification of algorithms can be given [Cristian 1991], based essentially on two criteria: *synchrony* and presence or absence of a *global clock*:

- Asynchronous systems that work without taking time into account at all, in the sense that no upper bounds of execution times of actions need to be given for the system to work correctly; thus the given algorithm is supposed to work under any timing constraints.
- Synchronous systems in which all processes can be considered as working off a common clock, for example digital circuits or distributed systems in which the clocks local to each node are synchronized (see, for example, section 9.6.6 for the low-level clock synchronization proposed in XPA).
- Synchronous systems in which a global clock is not available, but an assumption is made about the upper bounds of the time needed for actions, where these upper bounds refer to local time.

The systems in the first category are also called *systems with unbounded delay*. Those in the last two categories are *systems with bounded delay*. The verified protocols of the Delta-4 architecture are in the third category. General verification methods must be adapted to this class of systems, the main problem being the introduction into the model of a suitable notion of time.

The remainder of this presents the actual verification work in the Delta-4 project and the main results that have been obtained. Two annexes provide a more detailed description of the formal verification method by model checking. Annexe K introduces different models and possible notions of time used for verification; annexe L presents the nature of the service specifications and gives some formalisms allowing us to describe the specifications concerning time.

15.2.2. Verification of Protocols in the Delta-4 Project

The work mainly concerns the formal specifications of the considered protocols, the clarification of the assumptions made on the behaviour of the environment and the formal verification, from which the new implementations have been structured and developed. The primitive material for the formal verification activity in Delta-4 was an existing communication stack software, including the critical protocols to be verified, that was partially and informally specified. The results obtained have required tight interaction between the designer-implementer and the specifier-verifier teams.

The verification work has been carried out in two steps: first, providing formal specifications, then verifying formally their consistency.

The specifications of each protocol have been structured into three formal descriptions:

- A description of the state machines implemented in the protocol.
- A description of the properties of the environment (e.g., duration of the message transmissions, limitation of the number of messages lost in sequence, ordering of the delivery of messages). These properties of the environment correspond to the concept of "operational envelope" used in section 5.1.2.
- A description of the expected service, by a set of logic formulas. A classification of the usual properties is discussed in annexe L.

Formal verification has enabled detection of inconsistencies in these formal specifications and thus led to removal of design faults. Model checking techniques have been applied, using the *Xesar* tool [Richier et al. 1987] developed at LGI. These methods were chosen because they are suitable for finite state machines such as protocols and can be automated to a large extent, even for complex systems (the *Xesar* tool allows verification of quite large models [Graf et al.

1989]). Furthermore, they provide considerable help for the detection of errors since they are based on a complete search of the graph of all possible behaviours of the system to be verified, using appropriate algorithms. Error states can not only be detected, but execution sequences leading to them can be displayed.

15.2.2.1. Verification using the *Xesar* Tool. The *Xesar* tool implements model checking techniques: it evaluates properties given by formulas of the branching time temporal logic CTL [Clarke et al. 1986] or by Büchi automata [Büchi 1962] on a model generated from a program written in Estelle/R, according to its formal semantics. More details concerning the model can be found in annexe L.

The specifications of the system to be verified are described by a finite set of communicating processes, forming a *closed system*; that is, for each possible communication, the complete description of both emission and reception must be provided. Therefore, the *protocol environment* must also be described explicitly by a (set of) process(es).

Each verification is carried out on a model obtained from a program describing a particular system configuration embedded in its environment, with a particular initialization; such a program is called in the sequel a *scenario*.

The protocol description language Estelle/R is a variant of Estelle. The main difference is that in Estelle/R communication is modelled by the *rendezvous* mechanism. Communication by rendezvous requires explicit representation of finite buffers by processes. The problem arising from "communication through unbounded buffers" of Estelle is that certain deadlock situations cannot be detected, since it is possible to carry on filling some message buffer forever.

The formal semantics of an Estelle/R model is based on the ATP algebra [Nicollin et al. 1990] where a "clocktick" event is used for the translation of the "delay" construction. The global model is obtained by using the semantics of ATP for the parallel composition, i.e., interleaving semantics with synchronization of all processes on "clocktick" as discussed in annexe K.

Formulas of the CTL temporal logics or automata describing the properties that characterize the service specification, are evaluated on the model; the complexity of the model checking algorithms is discussed in annexe L.

An error is detected if, during the traversal, a state not compatible with the property is encountered. In this case, the execution sequence leading to this state can be analysed. If the traversal can be completed without detecting an error, the model satisfies the property.

If a complete traversal is not possible in a reasonable time, the absence of detected errors does not allow one to deduce that the protocol is correct. Nevertheless, it increases the confidence we may have in it. In this case it is also possible to make several partial traversals with different criteria for the choice of the successor state in order to increase the coverage, as proposed in [Holzmann 1990].

15.2.2.2. Formal Verification of Protocols. Applying formal techniques to perform a brute force global verification of the innovative Delta-4 protocols is an unrealistic task since their execution environment involves the whole Delta-4 architecture. The two main problems to be solved first are: the characterization of the protocol execution to be verified and the establishment of a suitable model for bounded time protocols.

Formal verification using the *Xesar* tool allows automatic analysis of *specific complex configurations* of the protocol execution. In each configuration, exhaustive analysis allows detection of all design errors, even those occurring in complex execution sequences and which are very unlikely to be detected by simulation or test, as they might occur with very low probability.

To infer the validity of the protocol in *any configuration*, one needs some "inductive" method. Different solutions have been proposed, generally based on an invariant characterizing the behaviour of the protocol. This invariant must have a "good" structure, i.e., it must allow induction on the structure of the configuration. However, the protocols considered in this project are too complex since they are based on the (simultaneous and successive) use of different paradigms (two phase protocols, election, reliable multicast, use of time out to avoid livelocks, ...). The complexity of the interaction of these paradigms does not allow description of the complete behaviour of the protocol by a "suitable" invariant.

However, for any particular protocol, non-explicit reasonings, based on considerations about the overall structure (e.g., making symmetry considerations or exploiting the splitting of the protocol into independent phases) provide convincing arguments that the general validity can be deduced from the verification of a limited family of configurations. For example, it can be argued that in the case of AMp protocols, configurations with three machines are sufficient, and that we can further limit the possible actions in any of these machines.

Practically, a finite set of scenarios (judiciously selected configurations of a fixed number of networked protocol machines) is exhaustively verified.

A crucial point in the verification is the choice of an adequate model, suitable in the present case to time bounded systems, such that all properties formally verified on the model must be true in reality. This adequacy depends deeply on the abstraction level used for the protocol machine description and on the modelling of its environment. For example, if the environment is modelled by a process "chaos" (i.e., that can modify transmitted messages in any possible way), all detected errors may not correspond to real errors in the protocol.

In the present work, the environment consists of adjacent layers of the communication stack; the model must take into account the different features characterizing the behaviour of these layers, such as buffering, occurrence of faults, and timing constraints.

The first two features are explicitly represented in the model. Concerning time, two kinds of modelling have been used, depending on the nature of the system to be verified:

- In systems where all message transmissions are implicitly clocked, timeless models are used. For example, in the Turbo-AMp or token-based AMp protocol, (an extension of the IEEE 802.5 token ring MAC protocol, see section 10.8.1) all messages are implicitly clocked by the token circulation mechanism. Therefore, the verification of a service property is made under some fairness assumptions, stating that the property is true provided the token is effectively circulating.
- In other systems, e.g., the token-less AMp (see section 10.8.2) and IRp protocols, the passing of time is introduced explicitly in the model, by means of a specific clocktick action. Thus, time bounds can be verified without using fairness assumptions, and some statements concerning time limits can be made. However, all the obtained results can only be guaranteed to be valid for the particular timer values for which the verification has been carried out.

A survey on time modelling for the purpose of verification can be found in annexe K.

15.2.2.3. Results of the Verification Work. Two families of protocols have been verified and consistent, structured specifications of both protocol machines and services have been provided:

- reliable group or multicast communication protocols (token-based and token-less implementations of AMp, see sections 10.8.1 and 10.8.2) — only the atomic quality of service (see section 10.4) has been verified;
- inter replica coordination protocols (IRp) managing the replicated aspects of the session service users, and supporting distributed fault tolerance by active replication.

For each protocol, formal specifications have been provided:

- The services have been characterized by a set of formulas, e.g., the AMp properties given in table 1 of chapter 10.
- All the environment assumptions have been specified. For instance, in the case of the token-less AMp protocol, this leads to the formal characterization of the abstract network given in table 3 of chapter 10. In the case of the IRp protocols, they consist of the atomic quality of xAMp service and additional properties stating that station failure indications are received in bounded time. Thus, network management does not need to be fully characterized.
- Only the abstraction of the protocol machine data structures relevant to verification have been defined. It is worth noting, however, that all transitions were in fact fully specified.

This task has been carried out in tight collaboration with the designer of the protocols because a detailed knowledge of the protocol architecture as well as of the verification method is needed. Some design errors were directly discovered during this collaboration.

The verification of the consistency of these specifications has been made as previously discussed. For example, in the case of token-less AMp, scenarios were grouped in different sets, covering each of the relevant steps of AMp execution, including: message transmission, monitor election, joining a group, leaving a group, and failure recovery. A justification for the chosen set of scenarios can be found in [Veríssimo and Marques 1990]. The "sub-services", i.e., the expected services for each of these steps, were also defined by sets of properties.

For instance, for the monitor election scenarios the following properties were defined:

- "There is one and only one winner of a monitor competition."
- "If the current monitor fails and there are still live members, another monitor competition takes place."
- "The monitor competition always terminates successfully unless all the group members fail."

Additional information is needed to define the scenarios associated with each step:

- The set of the possible initial states in which the steps can start. For example, for the monitor election step the initial states correspond to the states reached after the occurrence of an error in any other step: failure of the sender or of one of the receivers during a message transmission step; failure of one of the stations during a joining step; and so on. One or several scenarios are needed for each class of initial states
- The number of stations needed for the verification of each step. For example, for the monitor election step, it is possible to examine all possible error cases with three stations. The failed stations need not be represented.

Particular configurations with a fixed number (e.g., 3 for the token-less AMp) of interconnected protocol machines, covering critical cases derived from the structure of the protocol, have been verified with the *Xesar* tool. The generated models have reasonable size (on average 350,000 states for the token-less AMp). Non trivial inconsistencies, have been detected (and corrected) in the specifications.

Two types of inconsistencies have been identified: "superficial" ones that can be easily corrected, and "deep" ones that need more analysis to be corrected. Deep inconsistencies may show up inconsistencies in the application of the paradigms underlying the design of the protocol.

In both AMp and IRp protocols, a significant number of superficial inconsistencies were detected, for example: bad initializations of local variables; incorrect parameters; conditions in transitions that are too weak, etc.

A first example of a "deep" inconsistency was found in the token-less AMp: it corresponded to a livelock situation where, in some global states, a "monitor election" procedure can be restarted forever and will thus never terminate.

A second example is given by the IRp protocols.

Let us first describe shortly how these protocols work. The IRp protocols ensure that a family of replicated entities perform the same actions, and that an external user perceives all the replicas as a single one. Each correct replica has a complete knowledge of the state of the other correct replicas. To enforce this point, a replica uses the AMp protocol to inform the other replicas of any internal change.

When a message is sent outside the group of replicas, an election is carried out to select exactly one sender. Timers are used to control the relative speed of execution of the different replicas.

Three paradigms (atomic multicast, election, time-outs) are used together. As the number of possible behaviours is very large, conflicts occur. For example, two inconsistencies have been detected in the message send protocol:

- If a replica dies after its election as a sender, the other replicas are sometimes unable to elect another sender.
- If timers expire exactly at the moment a sender is elected, due to transmission delays, it can happen that the sender sends the message and a new election is performed, resulting in the message being sent twice.

Detailed results and the formal description of the services, of the protocol machines and of the environment assumptions can be found: for AMp in ([Baptista et al. 1990, Graf et al. 1989, Graf et al. 1990]) and for IRp in forthcoming reports.

To conclude, the benefit of this work is to force the definition of consistent formal specifications, that are of great importance when developing innovative protocols. Indeed, formal specifications do not only allow unambiguous characterizations of the protocol behaviour and facilitate reasoning about it, but some methodological points are to be stressed in the process of development for such systems:

- These verified formal specifications increase the confidence in a more correct design of protocols. As all possible behaviours are examined in a systematic way, the result is often more reliable than the result of mere simulation.
- The verification task and the definition of scenarios for verification require well-structured specifications; the software implementation can take advantage of this structuring of the specified distributed algorithm.
- The assumptions about the behaviour of the environment (for example the network in the case of AMp) are truly formalized.
- The service specifications are used for the implementation validation, for instance by fault injection (see next section).
- The automatic derivation from the formal specifications, of tests and assertions to be verified by the implementation can also be envisaged.

Note however, that in our experience, the automatic generation of executable code is not a realistic issue, in so far as the generation of *efficient* code depends highly on the characteristics of the environment in which the protocols are to execute. In the context of Delta-4, performance requirements do not allow automatic generation of code for significant parts of software, and

many local optimizations are required by existing heterogeneous components of the architecture.

15.3. Fault Injection

15.3.1. Introduction

The fault tolerance features of the Delta-4 architecture are based on the multicast communication system (MCS) that provides generalized multicast services. The proper operation of MCS is further based on:

- the verification of a set of well defined properties that characterize the extended service provided by the atomic multicast protocol (AMp),
- the assumed fail-silent property of the underlying network attachment controllers (NACs) hardware modules that connect the stations to the Delta-4 network.

The validation described in this section is thus aimed at:

- estimating the coverage provided by the fault tolerance mechanisms, which incorporate two levels of coverage:
  - the local coverage achieved by the self-checking mechanisms which control the extraction of the NACs,
  - the distributed coverage corresponding to the fault tolerance provided by both the defensive characteristics of AMp and the NAC self-checking mechanisms,
- testing, in the presence of faults, the service provided by AMp.

This validation is also intended to address the successive versions of the AMp software (obtained in particular as a result of the corrections induced by the design errors detected during this validation phase) as well as two distinct versions of the NAC hardware featuring quite distinct levels of redundancy (fail-uncontrolled NAC and fail-silent NAC)[2]. Both NAC architectures are made up of two boards:

- a *main board* that ensures the interfacing with the host computer,
- a *specific board* that connects the main board to the physical medium.

15.3.2. Fault-Injection-Based Experimental Validation

Fault-injection is particularly attractive [Chillarege and Bowen 1989, Crouzet and Decouty 1982, Damm 1988, Gunneflo et al. 1989, Lala 1983, Segall et al. 1988] as a complement of other possible approaches such as proving or analytical modeling. By speeding up the occurrence of errors and failures, fault injection is in fact a method for *testing* the fault-tolerance mechanisms (FTMs) with respect to their own specific inputs: the *faults*.

Basically, fault injection has thus the same characteristics and limitations as any testing approach: its accuracy depends heavily on the representativeness of the inputs of the test and its actual impact is related to the number of significant events — erroneous behaviours of the FTMs — observed during the test sequence. Stated in other words, the coverage estimates derived from a fault injection test sequence are usually easier to extrapolate to the real operational domain when they are low. As an example, if an unacceptably low coverage figure is obtained in the case of a test sequence where only permanent faults have been injected, it can

---

2 A *fail-uncontrolled* NAC refers to a NAC characterized by restricted self-checking mechanisms, while a *fail-silent* NAC is provided with improved self-checking mechanisms.

be confidently assumed that this figure constitutes an upper bound for the "actual" coverage, i.e., when more demanding cases (e.g., transient faults) are also considered.

Two important contributions of fault injection concern the verification of the FTMs and the characterization of their behaviour, thus enabling any weakness in their design and/or implementation to be revealed. Also, a statistical analysis of the responses obtained during the fault injection experiments enables some relevant dependability parameters — coverage, fault dormancy, error latency, etc. — to be estimated.

Different forms of fault injection experiments (e.g., fault simulation [Choi et al. 1989], fault emulation [Gérardin 1986], error seeding [Mahmood et al. 1984], mutation testing [DeMillo et al. 1978], physical fault injection [Côrtes et al. 1987], etc.) can be considered depending on i) the complexity of the system to be validated (the *target system*), ii) the types of faults injected and iii) its level of application at various stages of the development process [Arlat et al. 1990]. The fault injection method used here is the *physical fault injection* method: in this case, the faults are directly injected on the pins of the integrated circuits (ICs) that implement the prototype of the target system. Although this methodology can only be applied at the final stages of the development process, its main advantages are that the tested prototype is close to the final system and that it enables a global validation of a complex system integrating both hardware and software features of the fault tolerance mechanisms.

Another practical limitation that is often opposed to pin-level fault injection is related to the representativeness of the injected faults with respect to the internal faults (in particular in the case of VLSI circuits). Such a limitation can be realistically — albeit partially — overcome by the application of multiple intermittent error patterns on the pins. The multiplicity and the values of the applied error patterns can be either randomly generated or possibly deduced from a fault simulation analysis of the IC considered.

From a general point of view, the experimental validation is based on the concept of a fault injection *test sequence*. More precisely, a fault injection test sequence is characterized by an *input* domain and an *output* domain. The *input* domain corresponds to a set of injected *faults F* and a set *A* that specifies the data used for the *activation* of the target system and thus, of the injected faults. The *output* domain corresponds to i) a set of *readouts R* that are collected to characterize the target system behaviour in presence of faults and ii) a set of *measures M* that are derived from the analysis and processing of the *FAR* sets. Together, the *FARM* sets constitute the major attributes that can be used to characterize fully a fault injection test sequence. In practice, the fault injection test sequence is made up of a series of *experiments*; each experiment specifies a point of the *{FxAxR}* space.

15.3.3. The Testbed

For the application of fault injection to validate the Delta-4 architecture a distributed testbed was built using the fault injection tool *MESSALINE* developed at LAAS. The hardware testbed configuration is shown on figure 1.

The *Target System* contains four stations interconnected by the *Target System Network*[3]. Each station is made up of a NAC containing the implementations of the AMp and Physical entities that are under test together with a *Target System Host (TSH)* which activates and observes the service offered by the AMp. The activation consists in the generation of traffic flow through the target system network and the observation consists in the collection of data obtained from each station.

---

3 All the experiments carried out to date concern the 802.5 token ring version of MCS.

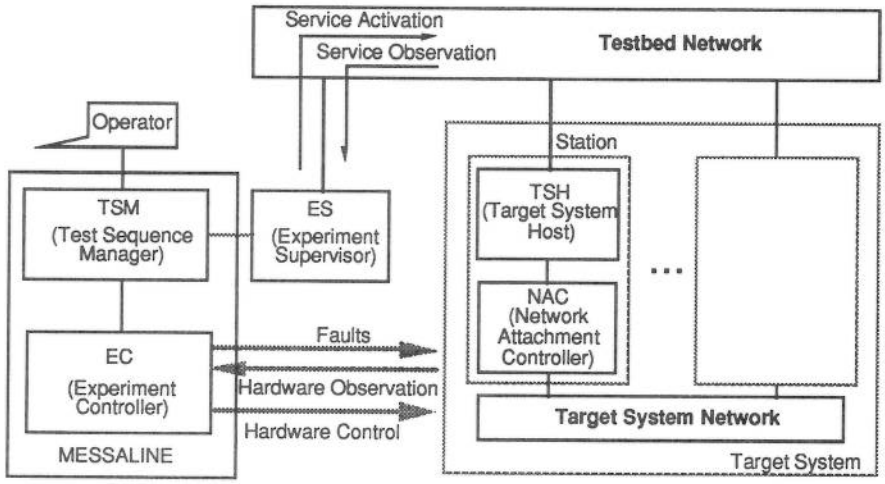

Fig. 1 - The Hardware Testbed Configuration

The purpose of the *Experiment Supervisor (ES)* is twofold: i) run-time control of the target system and ii) collection and analysis of the observed data. The *Testbed Network*, which is an Ethernet type LAN in the present implementation, ensures the communication between the ES and the TSHs. The ES is implemented on a Bull DPX2000 machine.

Most hardware and software necessary for the fault injection test sequence are part of *MESSALINE*. For sake of conciseness, only the two principal components are briefly presented here; a more detailed description of *MESSALINE* hardware architecture is given in [Arlat et al. 1990]

The *Experiment Controller (EC)* implements the injection of the elements of the *F* set and the collection of the hardware signals used to elaborate some elements of the *R* set. Faults are injected into the NAC component of a specific station (designated hereafter as the *injected station*) by the *forcing* technique[4]. A connection with the wiring concentrator of the target token ring network allows the states of the insertion relays of the stations connected on the ring to be read. These are used for the post-test analysis, as will be explained later. The only intervention of the operator is to position the probe on the circuit selected by the *Test Sequence Manager (TSM)*.

The TSM is implemented on a Macintosh II computer, connected to the EC and to the ES through serial lines. There are also physical connections between MESSALINE and all the stations (TSHs and NACs) to enable automatic hardware resets: the NACs are reset after each experiment whereas a host reset is requested only in case of a station crash.

In the reported experiments, the Target System is made up of four Bull SPS7 machines, running UNIX System V, as TSHs. The preliminary (fail-uncontrolled) NAC architecture tested so far contains only limited self-checking mechanisms, namely the mechanisms offered by the 802.5 token-ring standard (internal bus parity check, watch-dog timer, etc.). Improved self-checking (fail-silent) NACs, featuring duplicated processors and memory, that have been designed to interface a Ferranti Argus 2000 TSH are now being validated on the same testbed.

---

4 In the case of the *forcing* technique, voltage levels are directly applied by means of multi-pin probes on IC pins and associated equipotential lines.

15.3.4. Definition of the Test Sequence

This section presents successively the main parameters that specify the Fault, Activation, Readout and Measure *(FARM)* attributes of the test sequence.

15.3.4.1. The F Set. Faults are injected into a single NAC; the four stations were thus partitioned into two sets: the *injected station* (station S1) and the three *"correct"* (non-injected) *stations* (stations S2, S3 and S4).

For each IC, the injected pins, the nature and the timing characteristics of the injected faults were selected according to the order and form given below:

1) *Multiplicity (MX):* faults are injected on several (1, 2 or 3) pins of an IC, with a frequency of 50%, 30% and 20% respectively.
2) *Location:* selection of MX pins among the injectable pins, with a uniform distribution.
3) *Nature:* stuck-at-0 and stuck-at-1 faults, each with equal probability of occurrence.
4) *Timing characteristics:* the faults injected are essentially intermittent faults; their temporal parameters are composed of three values:
   - *lead time* (from start of experiment to first pulse): uniformly distributed between 1s and 40s,
   - *period:* logarithmically distributed between 10 µs and 30 ms,
   - *width:* uniformly distributed between 2 µs and 1 ms, with a duty cycle ≤ 50%.

In the lack of sound available data concerning actual IC failure modes, most of the parameters were selected according to a uniform distribution among range values selected quite arbitrarily. However, the limitation to a multiplicity order of 3 is to some extent substantiated by the results presented in [Gunneflo et al. 1989] for a microprocessor in the presence of single event upsets. These results show that more than 80% of the internal single-event upsets led to the occurrence of the first error pattern on 3 pins at most. The logarithmic distribution for the period is intended to obtain a significant number of short period intermittent faults while maintaining a wide selection range.

15.3.4.2. The A Set. The workload was varied to study its impact on the behaviour of the target system. The activation is characterized by the application of two types of traffic flows: *observed traffic* flow with respect to which AMp properties are tested[5] and *background traffic* flow to provide more realistic activation of the target system.

To provide a representative activation set, five activation modes have been considered for the stations; table 1 shows the transmitter and receiver allocations of the stations for each mode with respect to observed traffic and background traffic flows.

15.3.4.3. The R Set. Three types of readouts are collected for each experiment:

- *binary* readouts: activation of the injected faults, status of the ring insertion relays for each NAC, AMp properties derived from the analysis of the messages,
- *timing* readouts: time of activation of the injected fault, time of extraction of the stations,
- *message* readouts: number of messages exchanged for both traffic flows and number of messages positively or negatively confirmed for the observed traffic.

---

5 The observed traffic consisted of 100 messages.

Table 1 - Transmitter and Receiver Allocation per Activation Mode

T: Transmitter present, R: Receiver present, —: No Traffic, X: Not inserted

	Observed Traffic				*Background Traffic*			
Mode	S1	S2	S3	S4	S1	S2	S3	S4
1	TR	TR	R	R	TR	TR	TR	TR
2	R	TR	TR	R	TR	TR	TR	TR
3	—	TR	TR	R	TR	TR	TR	TR
4	—	TR	TR	R	—	TR	TR	TR
5	X	TR	TR	R	X	TR	TR	TR

The information concerning whether or not a particular fault becomes activated during an experiment can be obtained from specific monitoring devices implemented in MESSALINE that sense current variations on the pin(s) where the fault is injected [Arlat et al. 1990]. The status of these devices can be read by software. Such information was used to eliminate from the statistics those experiments where the fault was not activated during the experiment and to perform a direct measurement of fault dormancy for faults that were observed to become activated.

The collection of occurrence and timing readouts enabled empirical distributions to be derived for the fault dormancy. Empirical distributions were also derived for the coverage achieved by the hardware error detection mechanisms of the NACs and by the AMp.

On the average, each fault injection experiment took about five minutes. This large value is mainly due i) to the experiment set-up and ii) to the possible execution of the automatic recovery and restart procedures, in case of failure of the testbed after a fault has been injected. More specifically, the watch-dog monitoring the useful duration of each experiment was set to 110s.

15.3.4.4. The M Set. The measures considered for the analysis presented here consist of two types of measures: *predicates* and *time distributions*.

Let *E* designate the *error occurrence* predicate; i.e., E is true if the injected fault is activated on the faulted pin(s). Let I_i denote the status of the ring insertion relay of station S_i, i = 1, ..., 4; Ii is true if S_i is inserted into the ring.

The *local coverage* or *error detection predicate D* characterizes the efficiency of the NAC self-checking mechanisms:

$$D = E \bullet \overline{I_1}$$

D is true if the NAC of the injected station is extracted when the injected fault is activated (the expected behaviour in presence of faults)[6]. The notation: A • B is used to designate the conjunction between predicates A and B.

6 An opposite use of D has been made in the case of mode 5 to test if the station remained extracted when faults were injected; thus, in this case, a 100% coverage is assumed when the station remains extracted.

Let *P* designate the predicate corresponding to the conjunction of the subset[7] of the *AMp properties* considered (see table 1 of chapter 10 for a definition of the AMp properties). Let predicate *C* characterize the *confinement* of the fault/error (i.e., all the "correct" stations remain inserted in the ring). The *distributed coverage* or *fault tolerance* predicate *T*, that characterizes the defensive properties of the protocol at the MAC layer, can be expressed as:

$$T = E \bullet P \bullet C = E \bullet (Pa3 \bullet Pa6 \bullet Pa9) \bullet (I_2 \bullet I_3 \bullet I_4)$$

T is true whenever *all the AMp properties* are verified, the *confinement* of the fault/error is ensured when E is true. Although, it might *a priori* mask interesting characteristics, the grouping of the P and C predicates into one single predicate is substantiated by the two following remarks:

- the results obtained to date [Aguera et al. 1989, Arlat et al. 1989] have never led to an observation of P being false when C was true,
- the status of predicate P is of little interest when predicate C is false.

Thus, in subsequent analyses, the status of predicate T can be strongly related to the status of predicate C.

Two types of time distributions complement the analysis. The *fault dormancy* measures the time interval between the injection of a *fault* and its activation as an *error* at the point(s) of injection. If T_d denotes this random variable, and T_E and T_F the error and fault times respectively, then:

$$T_d = T_E - T_F$$

The *extraction latency* corresponds to the time interval between the injection of a *fault* and the *extraction* of the NAC of the injected station; if T_I denotes this random variable, and T_D the extraction time, then:

$$T_I = T_D - T_F$$

15.3.5. Major Results Obtained

15.3.5.1. Characterization of the Experimental Results. For each circuit submitted to fault injection, 30 experiments were carried out for each of the 5 activation modes. Accordingly, the complete test of one IC consisted of a run of 150 experiments that was fully automated to enable the experiments to be carried out overnight. A total of 40 circuits out of the 101 that compose the NAC (with restricted self-checking) was submitted to fault injection. Even though this represents only a subset of the circuits, the use of the forcing technique allowed a high proportion of the actual equipotential lines to be faulted, resulting thus in a pin coverage of 84%.

Another level of uncertainty is attached to the practical restriction to 110s of the observation domain; in particular, it is clear that a distinction has to be made between:

- a fault which does not become active during an experiment,
- a fault that would never become active.

However, it has to be pointed out that such an uncertainty may lead [Arlat et al. 1990]:

- either to pessimistic estimates (e.g. in the case of the local coverage predicate D),
- or to optimistic estimates (e.g., in the case of the T predicate).

---

7 The subset of AMp properties tested so far include *unanimity*, *non triviality* and *order*; testing properties such as *termination* and *causality* would require a global clock and global ordering of the observed interactions that were not implemented for the sake of simplicity.

Nevertheless, the analysis of the shape of the time distributions obtained provide an *a posteriori* means to support the acceptability of the observation domain. Furthermore, the exchange of several tens of messages (up to 100 messages) from the observed traffic flow during each experiment provides a sufficient activation domain enabling a reliable analysis of the P and C predicates to be carried out.

In this section, the focus is essentially on the presentation of summarized results concerning coverage estimations (local and distributed) and the evolution of three successive versions of the AMp. Detailed results and analysis of the influence of the activation modes, dormancy distribution, and others, can be found in [Aguera et al. 1989, Arlat et al. 1990, Arlat et al. 1989]. As the fifth activation mode is specific (faults are injected into a non-inserted NAC) and indicated a 100% coverage for the D and T predicates, the reported statistics concern the first four modes only.

15.3.5.2. Estimation of the Coverages. Figure 2 summarizes the major statistical results obtained concerning the estimation of the local and distributed coverages of the fail-uncontrolled NAC running the first version of the AMp software. The percents indicate the values of asymptotic coverage for the predicates E (error), D (detection at NAC level) and T (tolerance of AMp). The time measures indicate the means for fault dormancy and error latency distributions.

The minimum distributed coverage (NAC + AMp) is about 68 % (82 % x 83 %). The apparent fault coverage, i.e., the proportion of tolerated errors is about 85 % (17.5 % + 82 % x 83 %). The estimation of the actual coverage depends on the causes of the 17.5 % for the $\overline{D} \cdot T$ combination, which requires supplementary observations of the AMp.

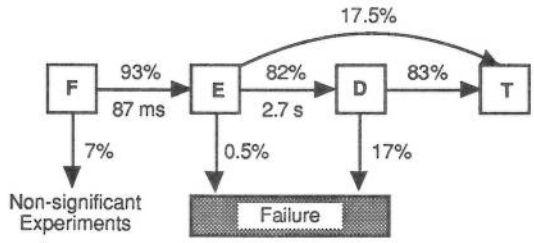

Fig. 2 - Summary of the Coverage Estimates

Further studies, including i) correlation tests and ii) the analysis of a supplementary set of 1600 experiments in which faults were injected directly onto the NAC relay control circuitry to ensure immediate ($T_I = 0$) NAC extraction have been carried out. These studies made it possible to identify the causes of the 14 % [82 % x 17 %] of errors correctly detected but not tolerated ($D \cdot \overline{T}$). About 8 % of these failures can be attributed to an erroneous behaviour of AMp. The major proportion (92 %) is probably due to the excessive latency of NAC self-checking mechanisms (this hypothesis is substantiated by observed correlation between the $D \cdot \overline{T}$ occurrences and measured detected latency [Aguera et al. 1989]).

15.3.5.3. Impact of Fault Injection on the Development Process. Traces and memory dumps recorded for each experiment in which non-confinement occurred provided the protocol implementers with useful data for the fault removal task. As a consequence, two more

versions of the AMp were submitted to the fault injection tests. For these tests, 8 of the circuits of the same NAC were selected among those that contributed most to the D• $\overline{T}$ proportion in the tests realized for the first version. The statistics presented were obtained in a set of 3600 experiments (8 x 150 x 3). The experiments showed that the distributed coverage was substantially enhanced, and that some types of errors were no longer obtained.

15.3.5.3.1. Evolution of the Distributed Coverage. Figure 3 illustrates the variation of the D• $\overline{T}$ proportion observed for three successive versions V1, V2 and V2.3.

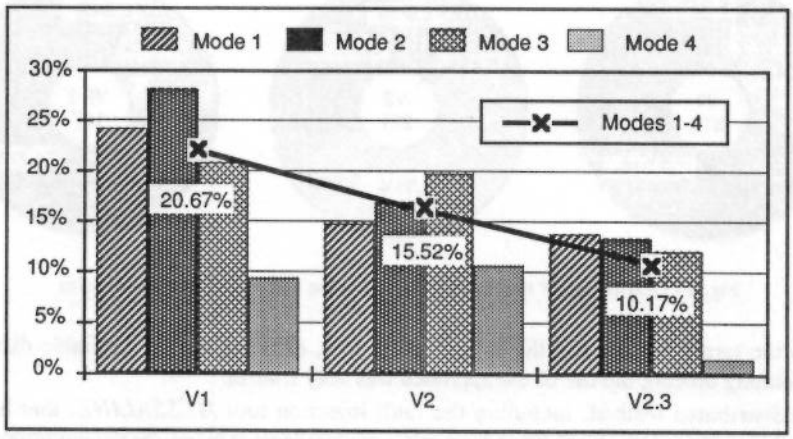

Fig. 3 - Variation of D• $\overline{T}$ Combination according to the Different Versions

It can be noted that passing from V1 to V2 had an impact on modes 1 and 2, although a certain degradation could be noted in mode 4. Instead, the passage from V2 to V2.3 caused an important decrease in the proportions of AMp failures, especially in modes 3 and 4. The percentages indicated on the histograms average the global variation for modes 1-4. The percentage of D• $\overline{T}$ is reduced by 50% from version V1 to V2.3, which demonstrates that there was an increase in AMp reliability.

15.3.5.3.2. Evolution of the Number of Errors per AMp Module. Figure 4 shows the distribution of the errors observed on the main software modules of the NAC for the three successive versions. For conciseness, only the modules in which errors were detected are shown. The *Monitor* operations enable the system to keep a coherent view of the multicast group in the network and to recover from station failures. The *Emitter* and *Receiver* entities perform the atomic data transfer operations. The *Driver* controls the exchange of information with the communication medium. A more detailed description can be found in [Ribot 1989].

15.3.6. Conclusion

In spite of the limitations of the physical fault injection approach (late application in the development process, for example) and the difficulties in applying it, especially in the case of distributed architectures (selection of faults to be injected; synchronization of the instant of fault injection with target system activity; parasitic mutations induced by the physical interference

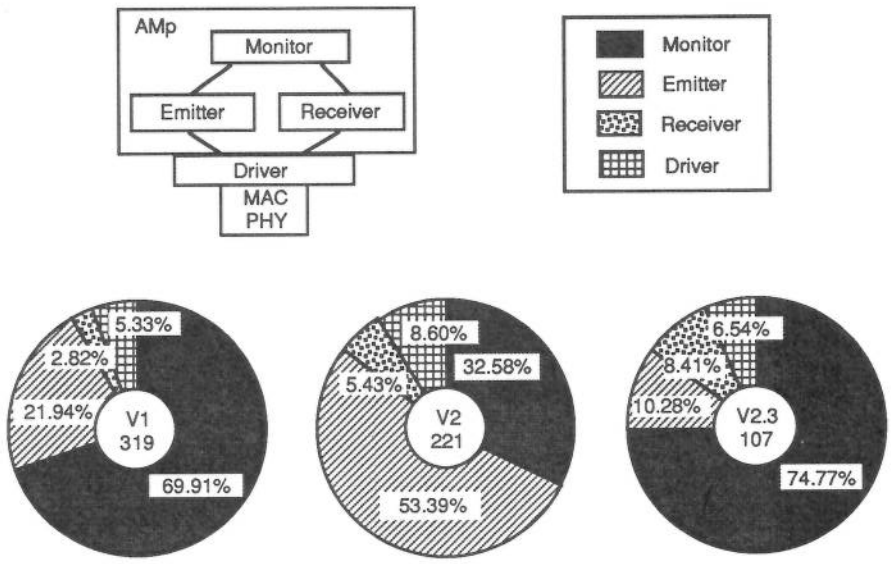

Fig. 4 - Distribution of the Errors observed in the Different Software Modules

between the target system and the experimental tool; effort to set up a reliable distributed testbed, among others), the use of the approach was very fruitful.

The distributed testbed, including the fault injection tool *MESSALINE*, that has been developed to carry out this work provides a fairly comprehensive experimental environment that enables:

a) a *global testing of the services* provided by AMp, which allows assessment of not only the interactions among several peer implementations but also those with the layers below,

b) the estimation of the effectiveness of the fault-tolerance features of the target system in the presence of *injected physical faults*,

c) the automated execution of a test sequence without operator intervention that was made possible only by the *integration of fault tolerance features in the testbed* itself. As an example, at the end of each experiment, should the injected station be found to have crashed, it is rebooted automatically.

For the sake of conciseness, only a fraction of the experimental results obtained has been included here. Other relevant contributions to mention are:

- the characterization of the behaviour of the system in the presence of fault, e.g., the impact of the activation modes; the influence of different types of faults (permanent, intermittent and transient faults); the impact of fault location (main versus specific board), among others;
- the identification of the limited performance of the self-checking mechanisms implemented on the tested NAC. Analysis based on a specific set of experiments showed that most AMp failure cases were caused by the excessive latency of the error detection mechanisms (especially for the main board). These results justify the need for the improved NAC architecture employing duplicated circuitry. The next step is thus the test of this NAC with enhanced self-checking (fail-silent NAC) to

evaluate local and distributed coverages to help to decide whether the benefits obtained justifies the increased cost.

The most recent version of the AMp software is being used now for the comparison of the two NAC architectures for the tests mentioned in the above paragraph. For these experiments, the property relative to the reconfiguration (called *consistent group view*, see table 1 of chapter 10) is included in the analysis. The preliminary results of these experiments are reported in [Arlat et al. 1991].

Finally, it is worth noting that the coverage estimates presented in this section correspond essentially to conditional dependability measures for the fault tolerance mechanisms tested (i.e., conditioned by the occurrence of a fault or an error). In particular, they do not account at all for the fault occurrence process. Work is currently being carried out that is intended to refine these coverage estimates by integrating the experimental results with a model-based description of the fault occurrence process. Towards this end, two approaches are currently conducted which are aimed at:

- weighting the coverages obtained for the faults injected on one IC by the failure rate associated to that IC,
- implementing the link between experimental measures and Markov models to derive dependability measures.

15.4. Dependability Evaluation

15.4.1. Introduction

Dependability evaluation is viewed here as quantitative rather than qualitative. Its aim is to compare the various design solutions, to define some essential parameters and to study their effects on system dependability. A global dependability model will be defined that can be used to evaluate several measures of the dependability achievable by different configurations of the architecture. The dependability verification using fault-injection will provide assistance in the necessary quantification of the model parameters.

The results obtained from the different activities of this work should provide the users of the Delta-4 architecture with guidance in the decisions concerning configuration of their own system. The final assessment of the quality of the architecture should help the development process itself as well as maintenance planning.

It should be noted that this activity is centred on the OSA architecture although it is possible that the models could be easily extended to cover the XPA architecture.

15.4.2. First Analysis

The aim is to develop a global model of the dependability of the architecture including both hardware and software faults. It will then be necessary to estimate the parameters of this global model and finally evaluate the dependability measures to study the reliability, the safety, the availability, etc., achievable by various configurations of the architecture.

It is very difficult to establish directly a correct global model; so a progressive method will be used. It is necessary to establish a global evaluation strategy in terms of inter-connected sub-models. After the study of the sub-models, it is necessary to aggregate them and study the global model. This organization in sub-models should also give some early feedback about the design of the different components included in the sub-models.

For the purposes of the evaluation, three levels can be distinguished in the Delta-4 OSA architecture:

- the hardware level (networks, NACs, hosts ...),
- the system software level (local executive communication stack, administration system, Deltase/XEQ),
- the application software level.

These levels will give a solution for the design of interconnected sub-models and simplify the dependability study.

We have defined three main objectives:

- modelling and evaluation of the communication hardware (comparison of the various communication topologies)
- extension of the communication architecture model to include the host-resident management information base (this model will include the MIB and MIB-management taking into account the replication of these entities on several hosts, before the extension to the complete hardware and software architecture).
- establishment of the global model of a target application and the evaluation of its dependability measures (aggregation of the previous sub-models in order to provide a framework for quantifying the dependability offered by a particular configuration of the architecture).

The first objective has already been achieved and the results will be summarised hereafter (see §15.4.4). The next step is to include the host-resident management information base.

15.4.3. Evaluation Method

Several methods for evaluating dependability measures can be distinguished: reliability block diagrams, fault-tree (or event-tree) analysis and state diagrams [Laprie 1983]. The main advantages of the latter are:

- their ability to account for the stochastic dependencies which result for instance from maintenance and solicitation processes, or from simultaneous consideration of several classes of faults,
- various dependability measures can be derived from the same model.

A state diagram is a graph in which the nodes represent the states of the system and the edges the elementary events leading to system transition from one state to another. The system model may be viewed as a representation of (i) the modifications of the system structure resulting from the events likely to affect the system dependability (fault-error-failure process, maintenance actions) and of (ii) other events of interest (e.g., solicitation process corresponding to the user's requests).

15.4.3.1. Markov Chains. When the elementary events can be considered as exponentially distributed (constant failure rates) the state diagram corresponds to a time-homogeneous Markov chain. Markov modelling is well adapted to dependability evaluation, it is well-suited for comparing different possible structures at the design phase (or during operational life if the architecture of the system allows this possibility) in order to select the "best" one.

Evaluating system dependability measures using Markov chain may be viewed as being composed of two tasks:

- model construction: derivation of the system behaviour model from the elementary stochastic processes,
- model processing: derivation of dependability measures from the system behaviour model.

The model may be very large for complex systems, in which case we need formal methods to construct it and program packages to handle it.

The current approaches aimed at formalizing Markov chain construction when accounting for stochastic dependencies are either algebraic approaches or graphic approaches.

Graphic approaches are based on stochastic Petri nets [Béounes and Laprie 1985, Beyaert et al. 1981, Molloy 1982] for which the basic idea is very simple, and thus attractive: when the transitions of the Petri net are weighted by hazard rates, the reachability graph may be interpreted as a Markov transition graph.

Advantage is taken of the SURF-2 dependability evaluation tool that is currently being developed by LAAS (independently of the Delta-4 project). This tool is based on Markov process evaluation techniques and model description can be carried out either directly using a Markov chain or using stochastic Petri nets.

15.4.3.2. Constant and Non-Constant Hazard Rates. The constancy of failure rates is a widely-recognised, and widely-used, assumption for element failures due to physical faults. Concerning the maintenance rate, it has been shown [Laprie 1975, Laprie et al. 1981] that considering corrective maintenance (repair) rates as being constant is, although not physically realistic, a perfectly satisfactory hypothesis for reliability evaluation. It is less satisfactory for availability evaluation since asymptotic unavailability may vary by a factor 1 to 2 when considering a constant repair rate or a constant repair time.

More generally, it can be considered that, under the assumption of exponentially distributed times to failures, assuming a constant hazard rate for the other processes is a satisfactory hypothesis as long as the mean values are small when compared to the mean times to failures.

Concerning software failure rates, due to the corrections introduced during the software life cycle, the failure rate generally decreases during the development and even in operational life. However, if no more modifications are performed or if the modifications still performed do not significantly affect the failure rate, consideration of a constant failure rate constitutes a good assumption. Usually this situation corresponds to an advanced phase of the operational life. Experimental results confirm this assumption [Kanoun and Sabourin 1987, Nagel and Skrivan 1982]. This failure rate (denoted residual failure rate) can be derived by applying a reliability growth model to data collected on the software in operation: the hyperexponential model developed at LAAS and used for several projects [Kanoun et al. 1988, Laprie 1984, Laprie et al. 1990] is the only model able to predict this measure.

What are the alternatives in terms of modelling techniques? There are three ways for handling non-constant hazard rates:

- time-varying Markov processes [Howard 1971],
- semi-Markov processes [Howard 1971],
- transformation of a non-Markov process to a Markov process by adding either (i) supplementary variables [Cox and Miller 1968] or (ii) fictitious states (the method of stages [Cox and Miller 1968, Singh et al. 1977].

Our recommendation (which has been put into practice for several years in the dependability group at LAAS) is the following [Costes et al. 1981, Laprie 1975, Laprie et al. 1990]:

- consider all the hazard rates as constant and derive a time-homogeneous Markov chain,
- perform sensitivity studies using the device of stages for those rates that are considered to be non-constant, starting with one fictitious state for each non constant rate, and stopping when addition of more states is of non-perceptible influence.

This recommendation stems from the following facts:

- a model is always an approximation of the real word, and this approximation has to be globally consistent,
- when modelling phenomena stochastically, the first moment generally determines the order of magnitude, the further moments bringing in refinements; an exponential distribution can be seen as the distribution corresponding to the knowledge of the first moment only.

15.4.3.3. Equivalent Failure Rate. When the non-absorbing states (non failed states) constitute an irreducible set (i.e., the graph associated with the non absorbing states is strongly connected), it can be shown that the absorption process is asymptotically a homogeneous Poisson process, whose failure rate (denoted equivalent failure rate) is given by:

$$\lambda_{eq} = \sum_{\substack{\text{paths from} \\ \text{initial state (I)} \\ \text{to the failed state}}} \frac{\prod \text{transition rates of the considered path}}{\prod_{\substack{\text{states in path} \\ \text{(except I)}}} \sum \text{output rates of the considered state}}$$

The reliability is then given by:

$$R(t) = \exp(-\lambda_{eq}\, t)$$

and the asymptotic unavailability is equal to:

$$\overline{A}(t) = \frac{\lambda_{eq}}{\mu}$$

where μ is the repair rate from the failed state.

Since the Delta-4 system is repairable, the associated graphs are generally strongly connected, and this approach will be adopted in the following: the different sub-systems will be evaluated through their equivalent failure rates.

15.4.3.4. Parameters of the Sub-Models. The parameters needed to establish the sub-models are (i) the failure rates of the different components of the architecture, (ii) the repair rates as well as the repair policy and (iii) the coverage factors that quantify the effectiveness of error-detection and fault-tolerance mechanisms. The estimation of these parameters will entail the use of failure rate data banks when such banks exist as well as the use of the results obtained from hardware fault-injection. In some cases, extrapolation of failure data obtained on similar projects will be of great help. Concerning the maintenance process, one has to assume some realistic repair policies (see subsection 15.4.4.1).

At the moment, reasonable figures have been guessed and the values of these parameters will be used before the end of the project to predict system dependability from a modular and parametric model. When the "real" values are evaluated from the definite project the corresponding parameters will be replaced in the model.

15.4.4. Modelling and Dependability Evaluation of the Communication System

The various hardware communication architectures are the 802.4 token bus, and 802.5 and FDDI token rings. In each architecture, every host possesses a NAC that interfaces the host and the underlying media. The communications software and a part of the administration software

are executed on the NACs. The couple host-NAC form a node or a station, and the set of the NACs with the underlying media constitute the communication system.

There are thus two essential aspects to be taken into account in the models: the communication topology and the nature of the NACs: fail-silent (with extended self-checking mechanisms) or fail-uncontrolled (with limited self-checking mechanisms).

15.4.4.1. Model Assumptions. We assume that a non-covered failure of one element leads to a total system failure and we will consider the following maintenance policy:

- a covered failure (of the NAC or of the medium) does not affect service delivery, moreover the repair of such a failure does not need service interruption,
- after a non-covered failure (of the NAC or of the medium), service delivery is interrupted, repair of all the failed elements is carried out before service is resumed,
- in case of one or several covered NAC failures, followed by a covered failure of the medium, repair priority is given to the medium,
- for the double ring, the wiring concentrators have the higher repair priority.

15.4.4.2. Notation and Numerical Values of the Parameters. The two types of NACs are modelled in the same manner. They differ by the numerical values of the parameters: for the fail-silent NAC, error detection coverage should be higher, the failure rate is also higher due to the greater amount of hardware necessary to enhance the self-checking. The equivalent failure rate of the communication system is evaluated as a ratio of the failure rate of the NAC. The double FDDI ring has the same model as the double 802.5 ring.

The main parameters of the models are:

- λ_N, the failure rate of the NAC — a value of 10^{-4} / h has been taken as a reference and corresponds to 1 failure per year,
- λ_{WC}, the failure rate of a wiring concentrator in the double ring, it should be about the same as the failure rate of a NAC (it has been taken in fact equal to λ_N in this study),
- n, the number of stations — fixed (arbitrarily) at 15,
- N, the number of wiring concentrators in the double ring,
- λ_B, the failure rate of the bus — a value of 2 10^{-5} / h has been taken; this corresponds to 1 failure per 5 years,
- λ_R, the failure rate of the ring — it has been taken equal to λ_B,
- μ, the repair rate, a repair duration of 2 hours has been adopted.

The coverage factors for the different elements are denoted: p_N, for the NAC, p_B, for the bus, p_l, for a link in the ring, p_{WC}, for the wiring concentrator and p'_{WC}, for the correct reconfiguration of the double ring after a non-covered failure of a wiring concentrator.

Let $\overline{p}_X$ be defined as: $\overline{p}_X = 1\text{-}p_X$ where $X \in \{N,B,l,WC\}$.

15.4.4.3. Summary of the Results. The expressions for the equivalent failure rates of the different architectures are very complex for the double media. However they can be simplified using the fact that: $\frac{\lambda_B}{\mu} << 1$.

The expressions of these failure rates considering only the first order terms are:

Single ring : $\lambda_{eq} \approx \lambda_R + n\,\overline{p}_N\,\lambda_N + n\,p_N\,\frac{\lambda_R}{\mu}\,\lambda_N$

Double ring : $\lambda_{eq} \approx 2\,\overline{p}_I\,\lambda_R + n\,\overline{p}_N\,\lambda_N + N\,\overline{p}_{WC}\,p'_{WC}\,\lambda_{WC}$

Single bus : $\lambda_{eq} \approx \lambda_B + n\,\overline{p}_N\,\lambda_N + n\,p_N\,\frac{\lambda_B}{\mu}\,\lambda_N$

Double bus : $\lambda_{eq} \approx 2\,\overline{p}_B\,\lambda_B + n\,\overline{p}_N\,\lambda_N + 2\,\frac{\lambda_B}{\mu}\,(n\,p_N\,\overline{p}_B\,\lambda_N + p_B\,\lambda_B)$

It can be noted that, for the single media the equivalent failure rates are limited by the failure rate of the medium and the non-covered failures of the NACs and that, for the double media, they are directly related to the failure rate of the non-covered failures.

The coverage factors are thus of prime importance (this has important consequences on the dependability of a system containing both Delta-4 NACs and non-Delta-4 NACs, see annexe M). However, due to the numerical values of the different failure rates the coverage of the NACs, p_N has more influence than p_B and p_I.

For the ring, duplication is worthwhile only for $\lambda_R > 5\ 10^{-5}$ / h even with a perfect coverage of the NACs, $p_N = 1$, (figure 5, for which $p_I = 0.95$ and $p_{WC} = p'_{WC} = 0.9$). This is due to the introduction of the wiring concentrators whose failure rates are of the same order of magnitude than the NACs. For instance, considering $p_N = 1$, duplication of the ring acts as follows:

- for $\lambda_B = 10^{-5}$ / h, it increases the unavailability from 11 min. to 48 min. / year,
- for $\lambda_B = 10^{-4}$ / h, it decreases the unavailability from 1 h 45 min. to 58 min. / year.

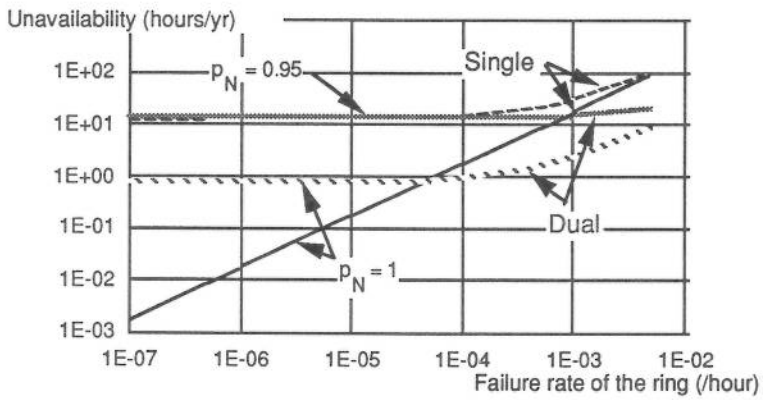

Fig. 5 - Communication System Unavailability (in hours per year) for the Single and the Double Ring, with $p_N = 0.95$ and 1.

With respect to dependability improvement due to the duplication of the bus, the unavailability of the communication system with a single and a double bus, versus the failure rate of the bus (λ_B) and for $p_N = 0.95$ and 1, is given in figure 6.

When the coverage factors are less than 1, duplication can lead to dependability deterioration depending on the value of the bus failure rate: for instance for $p_N = p_I = 0.95$,

improvement is effective only for $\lambda_B \geq 2\ 10^{-5}$ / h. This is because when λ_B is low, dependability is conditioned by the NAC failures.

When the coverage factor of the NAC is equal to 1, duplication is worthwhile, for example:

- for $\lambda_B = 10^{-5}$ / h, duplication of the bus decreases the unavailability from 0.2 (11 min. / year) to 1 min. / year,
- for $\lambda_B = 10^{-4}$ / h, unavailability is decreased from 1 h 45 min. to 11 min. / year,

which is a significant improvement.

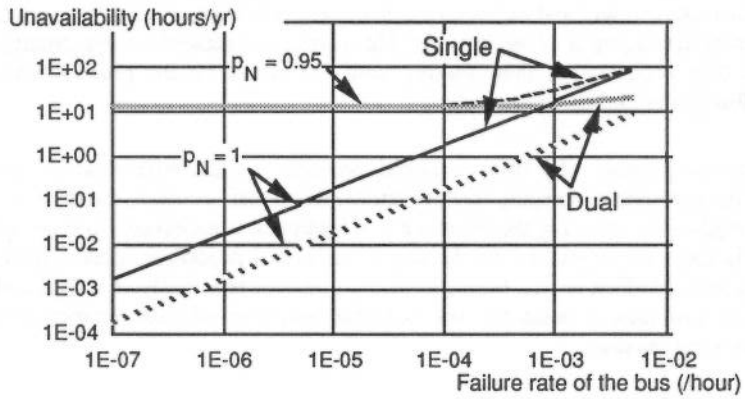

Fig. 6 - Communication System Unavailability (in hours per year) for the Single and the Double Bus, with $p_N = 0.95$ and 1.

Concerning comparison of these architectures, it is very difficult to classify them. Assuming the same failure rate for the bus and for the ring leads to the same expression of the equivalent failure rate and for the unavailability of the single medium. Figures 5 and 6 show that for reasonable failure rates ($\lambda_B \approx \lambda_R < \lambda_N$) dependability measures are independent from this failure rate. Which means that the single bus and the single ring are equivalent.

Duplication of the medium can even deteriorate the dependability measures depending in the parameter values. The results enable the different architectures to be compared according to the various parameters in order to make a tradeoff and to select the more suitable architecture. For instance, for the considered values, the double bus seems more interesting than the double ring for $\lambda_B < 4\ 10^{-3}$ / h, however, the value of the failure rate of the wiring concentrator is of prime importance: a lower value of λ_{WC} acts in favour of the double ring.

15.5. Software Reliability

15.5.1. Introduction

The idea of using the collection of data to carry out a reliability evaluation is not a new one. The techniques have been applied in several areas, including hardware reliability evaluations, over several years. Data collection for software has also been successfully implemented in several projects and was introduced within Delta-4 as the basis for both quantified and qualitative evaluation. This section provides a general overview of the reasons for data collection, its

relationship to software testing and software reliability prediction and a discussion of the problems of data collection. It then goes on to describe the data collection and reliability modelling activities undertaken within Delta-4 and to discuss some project-specific topics.

15.5.2. Reasons for Data Collection

There are several reasons for collecting data. These can be broken down into three separate levels of interest as below:

15.5.2.1. Managerial. At the highest level, data collection allows managers to have a clear picture of the progress of the development of the software and the problems encountered. It aids in overall project control and increases the amount of information available to management during the actual development of the software. This gives management the opportunity to solve problems as they occur rather than waiting until the effects of the problems have been propagated throughout the rest of the project.

15.5.2.2. Development. The data collected provides a clear historical record of the evolution of the software by keeping track of the incidents that occurred, the faults found and the changes made, thus allowing the developers to backtrack if necessary. In cases where the information is analysed as part of the development control procedure, data collection may prevent duplication of effort arising from addressing problems previously identified and solved. The data also provides a basis for the sensible planning of maintenance and future enhancements to the product.

15.5.2.3. Research. The data collected can provide the information necessary for further research into means of achieving and predicting software reliability using static analysis tools and modelling techniques. Significant research has already been done on the modelling side but work on the use of static analysis tools and their effect on reliability has so far been limited to a very small section of the software community. This research in the long term should benefit both management and software developers by enabling managers to locate problems earlier in the project and developers to have the opportunity to gauge how well the module has been constructed, before any testing has taken place. Frequently, extensive maintenance can prove to be detrimental to the structure and complexity of the code. The application of static analysis tools during routine maintenance and enhancements should allow the effects of the changes made after initial development to be assessed.

15.5.3. Planning and Implementation of Data Collection

To ensure that the data collected is of high quality, the planning and organization of the collection activity should take into account the need for adequate resources and skilled staff. It is also desirable to automate the data collection as much as possible to improve consistency and validity and to establish a project contact within each participating company. This contact's duties should include the management of data and the data collection activity to ensure the completeness of the data and data set, with an emphasis on accuracy and quality of data. The contact will also be able to act as an intermediary between those storing and analysing the data and those collecting it.

Some projects have taken the approach of collecting as much data as possible, as it may be of potential use in the future, although at the time it would serve no purpose. Given endless resources, time and man-effort, this may be a very useful thing to do. However, collecting data incurs cost and the financial constraints placed upon the project will dictate the depth and range

of the data collection activity. With this in mind, some thought should be given towards the most cost effective means of collection.

Some data can be accumulated for little cost, e.g., by use of static and dynamic analysis, whereas collection of other data can be more expensive. Static analysis tools provide valuable data in their own right but, the very fact that they are static analysers means that the code is not executed, and hence they cannot provide dynamic data, e.g., test coverage, time between failures.

The data collected should be "multi-purpose" to enable the use of the original data in a variety of areas. For some applications, minor amendments may be necessary but it is important to ensure that these amendments do not violate the integrity of the data.

On the research level, the data collected can be used both in a reliability evaluation and to assess the effectiveness and correctness of other techniques. Perhaps the best way to illustrate this point is by an example — several sets of data are input to a new software engineering tool that predicts very high failure rates. If we already know that the data has come from code that has been highly reliable in operational use over a considerable period of time, then confidence in the tool will be significantly reduced.

The data can also be used as a yardstick for project planning and projections on a managerial level and as a basis for future work on similar projects on the development level. It provides developers with the opportunity of learning from their mistakes and achievements by applying the data to the identification of their main problem areas or strengths, hence potentially improving the quality of their work.

15.5.4. Data Collection and Software Developers

To gain the best results from any data collection procedures, management should be aware of the reasons for data collection so that they can appreciate both the need for data collection and the need to motivate those people actually collecting the data. The collection mechanism should be made as simple as possible to ensure that the data collected is of an acceptable quality. Once the data collection methods are produced and agreed, they should be disseminated in the most appropriate manner.

Feedback of the results obtained on the data should be given at regular intervals in a clear and concise manner. Individuals are generally keen to improve the standard and quality of their work and are therefore interested in receiving feedback as long as it is done constructively. If the data collection activity is perceived in any way to be a means of performance measurement then the developers will be demotivated and the data collection activity will fail [Littlewood 1987]. It is important that this feedback is provided as quickly as possible to improve co-operation, even more so in cases where the staff involved in the software design and development are also responsible for testing.

The analysis of the data collected may allow software developers to discover answers to some very important questions:

- *Am I always making the same sort of mistakes in the same areas?*

  Records will indicate whether or not there are constant sources of error affecting the quality of particular products.

- *How effective are my testing procedures?*

  Analysis of the records may show that the testing itself, rather than the items tested, tends to be at fault. This can lead to a more effective testing practice.

- *How do I know what to test?*

The results of any walkthroughs that have been done allow developers to pinpoint any areas of weakness inherent in the code. Static analysers provide the developers with an internal picture of the data or control flow within the modules. This allows the developer to produce test data that will address the areas of weakness and increase the structural coverage of the code.

- *How do I know when to start testing?*

  By using walkthroughs as well as static analysers it is possible to significantly reduce the time spent in finding those faults that can be identified before the dynamic testing activity begins. Dynamic testing should begin only once there is a significant decrease in the cost-effectiveness of other techniques and the developers are satisfied that errors already identified have been removed.

- *How do I know when to stop testing?*

  The data collected provides an overview of the number of faults found and allows these faults to be classified. It cannot provide a hard and fast rule on when the testing should be stopped but can give an indication of the effectiveness and coverage of the testing already completed.

It should however be reiterated that while it is of the utmost importance to encourage the data collectors to proceed in a diligent and professional manner, so that the best results possible are attained, their own management needs to be persuaded about the importance of what they are doing. Without the approval of management, the data collection procedure will not be given the priority it requires.

15.5.5. Data Collection and Project Managers

Managers of all software projects need to have access to information on the development of a particular piece of software. They need to be able to collect and review operational information in order to have the fullest possible picture of the current project. The data collected as part of a reliability evaluation provides additional information on 'good' and 'bad' trends in performance, suspected or actual deviations from experience of previous projects, as well as providing input to future projects to help support:

- feasibility studies,
- project planning,
- project costing,
- resource allocation,
- choice of development methods, testing methodologies, process control and support facilities.

This enables managers to do their job more efficiently and make cost-effective use of the resources available.

Data collection must be purposeful. It should avoid obstructing or duplicating other activities, and if possible be built in to the development process, in order to make it acceptable to project staff. In many cases, those involved with the project may not fully appreciate the potential future benefits from regular data collection and may be tempted to give it a low priority at times when it should have high priority, e.g., during testing. In addition, the short-term benefits of data collection and analysis may not be immediately apparent as it takes some time to gather a suitable subset to make any reasonable statistical comparisons. However, the benefits of some data collection may be immediately available if the data required for such a subset is obtainable for other sources or there are commonly accepted precedences, e.g., McCabe's complexity measure [McCabe 1976].

For the best possible results, it is necessary to start data collection as early as possible in the project but this will vary from project to project. Data collected at a very early stage during testing should provide information on the progress being made by developers of individual modules, allowing managers to monitor closely those modules that are critical to the development of the system. Thus, a greater managerial control can be exercised over the entire software life-cycle.

15.5.6. Data Collection and Testing

Before the testing of any software product, it is very important to have produced a test plan defining the testing work in detail. This plan should provide a baseline for the work to be done and will help to provide effective status reporting. Data collection should be seen as an integral part of the testing activity and provides an ideal medium for the recording of the test results.

If data has already been collected before the start of the testing activity the results obtained can be used to indicate those modules that are more likely to contain faults. It is then possible to test these "faulty" modules first so that any modifications required can be made while the rest of the system is being tested. Using data previously collected from similar projects it may be possible to predict roughly the number and type of faults inherent in the software. Combining this prediction with the number of errors actually detected can provide an indication of the success of the testing activity.

The approved ANSI standard [IEEE 829] for software test documentation describes a test log and a test incident report and concentrates on providing a framework for collecting data during dynamic testing. The ANSI test log describes a chronological record of relevant details about each test performed, and the subsequent results (error messages generated, aborts, requests for operator action, etc.). Discrepancies, including what happened before and after any unexpected events, are described.

The ANSI test incident report documents any event that requires investigation or correction, summarising the discrepancies noted and refers to the appropriate test specification and test log entry. It includes the expected and actual results, the environment, anomalies, attempts to repeat, and the names of the testers and observers. Related activities and observations that may help to isolate and correct the problem are also included [Hetzel 1984].

15.5.7. Data Collection Problems

All data collected must be accurate and of the necessary level of detail so both developers and analysts can understand, and in some way measure, the software development process. Often the inconsistency of data definitions and a lack of commonality of terminology impose barriers to the general usage and effective interpretation of the available information.

The ANSI standard on Software Test Documentation described in the previous section goes some way to eliminating the problems encountered with the software data collection process, by providing collection standards. However, for data collection in general, there is still a need for commonly accepted terminology and data definitions. This would provide a solution to some of the common data collection problems listed below:

1. Inconsistent Definitions:
    - Inconsistent phase definitions.
    - Using terms without actually defining them.
    - Inconsistent use of terms which have standard definitions.
    - Standard definitions that do exist may not be freely available.
    - Accepted definitions that are contradictory.

2. *Observational Bias*. The problem with any subjective measure is that for a single given condition people will supply different answers. Some people are fundamentally very generous or critical when making subjective judgements. In other cases, ratings are conditioned by the individual's desire for the project to succeed, or at least appear to be successful.
3. *Local vs. Global Frame of Reference*. Consider a small department that consists exclusively of highly experienced and talented personnel who have been using a given development method for several years. If they are asked to rate the level of rigour with which they have applied that development method on a particular project, they may rate it as medium rigour with respect to the rest of the department's projects. However, with respect to other departments within the same organization that have limited experience of the development method, they should be rated as applying a high level of rigour.
4. *Averaging and Side Effects*. When project or subsystem assessments establish ratings of complexity, required reliability, timing constraints, etc., they tend to provide the rating for the most highly-stressed portion rather than the average rating across the project or subsystem. Also, people will intuitively give large software projects higher complexity ratings than small projects. This can be avoided by using tools such as COCOMO (COnstructive COst MOdel) [Boehm 1981] and several of the static analysers to calculate intrinsic complexity
5. *Double Counting*. In cases where a particular piece of code is being developed for use in two separate subsystems the information on the software may be included twice.

During the data collection process, problems will usually arise concerned with the accurate recording of software metrics. The values collected ought to be:

- *Repeatable* — two independent data collectors would obtain the same value if they were to measure the same item.
- *Comparable* — the metric values obtained from different items have been obtained using the same procedures or an equivalent translation process has been produced.
- *Verifiable* — the values can be checked for clerical errors and inconsistencies.

To satisfy these requirements, it is necessary that data collection procedures be implemented and adhered to, since the way in which data is recorded, verified and analysed will influence the success of any data collection activities. The following suggestions will however ensure that the data collection is as successful as possible:

- data collection should be integrated into the development process
- data collection should be automated wherever feasible
- data should be treated as a company resource and facilities should be available to keep historical records of projects, as well as monitoring current projects
- data that cannot be collected automatically should be collected at the time, not based on the recollection of past events, and verified immediately
- software engineers should be motivated enough to keep accurate records, and to participate in the data collection process
- the timescales between data collection and analysis should be minimized

A data collection activity following some, if not all, of the suggestions above will benefit the project as a whole and provide important information for use during reliability evaluations.

15.5.8. Data Collection and Delta-4

Within Delta-4, consideration was given to the method of data collection, its use within the project and the definition, storage and analysis of the data. This helped to remove ambiguities in the results of the collection caused by poorly specified requests for information and therefore improved the validity of and confidence in the data. The data collection manuals used attempt to either define the terms used or provide examples in cases that are open to misinterpretation.

As it was also important to reduce the costs of data collection to a minimum, the most cost-efficient techniques were implemented wherever possible. However, because the reliability growth models being used for the Delta-4 reliability evaluation require inter-failure times, it was necessary to accept a slightly larger overhead in order to be able to provide the data to drive the models.

Unfortunately it was not possible to introduce data collection until most of the code had been written and, in most cases, some initial testing had already been undertaken. It was however envisaged that sufficient data would be collected to allow a detailed reliability evaluation to take place during the project.

The original data collection activities that were produced ranged from extremely detailed collection activities to a set of simplified data collection forms. The detailed collection activity that was to be carried out at Ferranti on the development of Deltase for Ada consisted of a comprehensive set of collection forms as outlined below:

A) An operating environment form containing information on the hardware and software resources in use at each development and/or test site.

B) Several forms that only need to be filled in once for each product (e.g., Deltase for Ada). These forms provide background information and, although they are not directly used in the reliability evaluation, deliver a basic description of the structure of the software product and process. They are:

- Product form — contains information on the components and tasks associated with the product.
- Task structure form — contains information on the relationships between tasks
- System/subsystem structure form — contains information on the relationships between the different software components and how they are integrated.
- Textual system/subsystem structure form — contains information on the relationships between the textual components of the system.

C) Two forms that need to be completed for each task:

- Task definition form — contains information on the type of task being performed and dependencies between tasks.
- Task resource form — contains information on task duration and effort expended. This has been separated from the task definition form to ensure data confidentiality.

D) A form that needs to be completed for each component. The form required differs depending on whether the component is a software component or a textual component:

- Software component form — contains basic metrics and cross-references to the system/subsystem form and the task form.
- Textual component form — as for the software component form.

E) A final set of forms that provide the data required to drive the reliability growth models described later. This data is collected from several phases of the development process — in the early phases from design and code walkthroughs, in the later phases from dynamic testing and finally during maintenance. These forms are all cross referenced to allow the data to be processed into the correct format for the models.

- Test history — contains information on the behaviour of the software under test.
- Incident form — completed for each incident. Contains information on the cause and severity of the incident.
- Fault form — completed for each fault. Contains information on how the fault was discovered, when it was actually introduced (i.e., during which phase of development) and its cause.
- Change form — completed for each change. Contains information on how the need for change was determined and the purpose of the change (e.g., planned enhancement, fault correction).

The use of several forms at different levels of detail was designed to provide a comprehensive picture of the software being developed and to ensure the completeness of the data set over time. However, due to a change of direction in the project out of the control of the data collectors, this detailed data collection was not implemented.

A simplified data collection manual has been provided to other Delta-4 partners who were willing to collect data but did not have the time and effort available for an extensive data collection activity. This manual consists of only three forms — a software log, an incident report and a change report. The software log provides a detailed history of the execution of the software over time and includes details of the test phase and test type for data collected during testing. An example of the form is shown in figure 7.

This form is, however, not restricted to the testing phase but could be used throughout software development. The other forms provide more detailed information on the incidents and changes referenced in the software log and are very similar to the incident and change forms used in the more extensive data collection activity. Using this simplified approach minimizes the time spent on data collection but still allows sufficient data to be collected to be able to apply the reliability growth models to the software.

The forms in the simplified data collection manual are compliant with the ANSI Standard for Software Test Documentation described earlier. However certain areas of the standard are not necessary for the Delta-4 data collection activity. Where there is a mapping between the needs of Delta-4 and the standard, the two have been kept as consistent as possible.

Within Delta-4, to complement the work done on data collection, the mechanisms for data storage, transfer and analysis were set up as shown:

- *Data storage*. All data collected has been stored in a central analysis database and consideration has been given to the content and structure of this database. Plans were made to validate and verify the data before entry, incorporating the lessons learnt from previous data collections. A consistency checker is available to ensure that the data sets within the database are as complete as possible.
- *Data transfer*. Applications have been developed to execute some static analysis tools in a consistent manner and output the results to standard flat files. This standard flat file format allows the data to be fed directly into the central database.
- *Data analysis*. The data in the database can be output in the format required by a generic statistical package. The analysis process has been automated so that output

from the static analysis tool can be fed into the statistical package and analysed. A list of the code metrics is then automatically produced that indicates which metrics are outliers.

SOFTWARE LOG

DATA COLLECTOR : _____ COMPONENT ID : _____ VERSION ID : _____

DATE	START TIME	END TIME	TEST PHASE	TEST TYPE	INCIDENT ID	CHANGE ID

Fig. 7 - The Delta-4 Software Log

Although data collection could not be introduced right at the start of the project, every effort has been made to ensure the effectiveness and success of the data collection activity by providing automated versions of the forms and keeping the collection as simple as possible. The implemented mechanisms allowed prompt and accurate feedback directly to the relevant project members.

15.5.9. Data Collection and Predictions

Continuous data collection means that it is easier to statistically identify trends as and when they occur. The summary statistics provided as a result of statistical analysis should include information on the most frequent error causes, the most common means of detection and the most error prone modules. Detailed statistical analyses should also be provided for any trends that give cause for concern.

The reliability of software can be qualitatively and quantitatively assessed during its development, provided the use of the software is restricted to its specified purpose. Data collection can lead to an improved understanding of the software development process and this improved understanding extends to the reliability assessment of the software product itself. There is currently no single agreed method of producing reliable software and therefore, the collection, storage and analysis of data can aid in the evaluation of the links between the development method(s) and the reliability of the software.

Software reliability models use data on the times at which software failures are observed to occur, although the data can be processed for input to models that require data about the number of failures within a given timescale. For use in driving the software reliability models, the failure data will need to be processed, as several failures may result from one error.

All the reliability growth models rely on inter-failure times and require that a minimum number of errors occur before the data can be fitted to the model. Once the data has been processed, the use of a general linear model may help to identify problem areas in the software. When enough error data has been generated, the data is run through the reliability growth

models. Unlike hardware reliability modelling, where large quantities of a particular piece of equipment can be tested, software reliability modelling involves just one unique software product. The profile of the software changes when each error is encountered and repaired, with the result that it is impossible to determine in advance which model will best fit the data. Not only is the piece of software itself unique but so is each operational environment in which it is to be used. Therefore an assessment of a piece of software will contain restrictions as to the applicability of the reliability attached to the software.

A considerable number of reliability growth models have been published over the past twenty years and there has gradually developed an underlying unity in the approach of the models to the problem of software reliability [Mellor 1987], in that they adopt the following assumptions:

- the system contains a set of errors on delivery,
- each error causes failure independently of all the others,
- each error causes failure at its own particular rate,
- failures due to the manifestation of a single error occur as a Poisson process[8].

Some of the models also assume that on manifestation, an error is immediately and perfectly removed.

A software suite of nine models is currently produced by Reliability Consultants Ltd. and can be used to fit data to the models and to enable an analysis of the predictive accuracy of the models for each data set.

To use these models, the following procedure is implemented:

- assume that there has been a sequence of failures during continuous use of the software
- collect at least the minimum number of processed failures required to run the model and the times between failures
- collate the inter-failure times of the software, $t_0,.....t_n$
- verify that the chosen model fits the data reasonably well
- take the chosen model and fit parameters using statistical estimation techniques
- use the fitted model to estimate quantities of interest, e.g., current failure rate, time to next failure, number of remaining errors, etc.

The technique used to analyse the models ([Rook 1990], chapter 6) is that of partitioning the processed failure data so that the observed data on the first failures $t_0.....t_{i-1}$ are fitted to the models and the predictions made by the models when the data have been fitted to them are compared with the subsequent failure data $(t_i,....t_n)$ thereby giving a measure of the predictive accuracy of the models. The accuracy of the predictions is analysed using standard statistical techniques, such as u-plots and prequential likelihood, which will determine bias and noisiness in the dataset.

The testing scenario should match, as closely as possible, the environment in which the software will be put to use. To this end, records of all data input during testing should be kept, so that comparisons of testing and the operational profile can be made. It is very important that

---

8 If a number of incidents are recorded within a set of time periods, and the following three conditions are satisfied, then the process is a Poisson process:

a. Events occurring in two disjointed time intervals are independent
b. The number of events occurring in a time interval (t_1, t_2) is only dependent on the length of the interval $t=t_2-t_1$ and *not* on t_1
c. The probability that more than one event will occur at the same time, and the probability that an infinite number of events will occur in some finite time interval are both zero.

the data input during testing is carefully selected to optimize the test coverage of the paths through the software that will be exercised in operational use. Figure 8 shows one possible relationship between the set of inputs leading to failure and the set of inputs covered by testing.

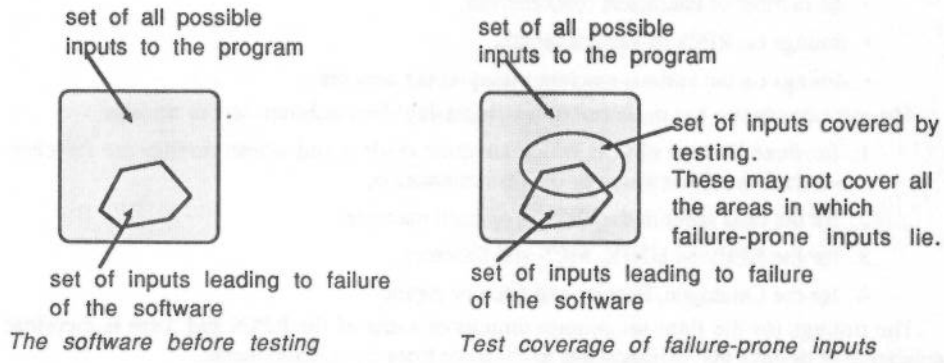

Fig. 8 - The Software Before and After Testing

There is an important potential problem in the simulation of the operational profile. A relatively small change in the characteristics of the profile can often lead to the execution of a previously rarely used piece of code, resulting in a whole new cluster of errors being found. Conversely, a larger change in the profile can have little or no effect on reliability. This is counter intuitive, but a corollary of this property is that a small error in determining the operational profile can cause a massive error in determining the reliability ([Rook 1990], chapter 1).

15.5.10. Predictions in Delta-4

The suite of nine models discussed above is being used to evaluate the reliability of the software. However, since Deltase is a support environment, this has raised several points of interest:

- The support environment does not execute continuously and has been designed to supply many services by multiple and diverse mechanisms in such a way as to minimize overhead. As a result, the measurement of execution time is difficult to achieve and an instrumented version of the code has been produced to provide execution time measurements. It is probable that the presence of the instrumentation may affect the results obtained since the process of measuring will affect the quality of the measurements.
- The distributed architecture means that ports on multiple different machines are neither fully independent nor fully interdependent. This has implications for the timing measurements, as different machines have different timing capabilities.
- The replication of software components raises problems in the processing of the data for the models. If the times of their parallel executions are summed, this will give an inaccurate measure of the reliability, since it is the overall reliability produced by the system architecture that interests us, not the duplicated reliability measurements of the replicas.

The instrumentation inserted into the code has entailed a prolonged effort to analyse the data produced by the instrumentation. Times for the remote procedure calls (RPCs) have had to be extrapolated from cumulative figures for:

- the number of RPCs sent/received,
- the number of characters sent/received,
- timings on RPCs of various length,
- timings on the various machines used on the network.

The instrumentation has produced (after processing) four different sets of timings:

1. for three Deltase objects which are quite distinct and whose timings are therefore not duplicated elsewhere by the instrumentation,
2. for the time spent in the libraries on each machine,
3. for the RPCs on UNIX, MCS and Ethernet,
4. for the Catalogue, Factory and Factory Agent.

The timings for the final set contain timings of some of the RPCs and there is therefore duplication of timings that is impossible to calculate from the measurements.

To enable a meaningful analysis, the software problem reports have had to be cross-referenced to the incident report forms generated by the instrumentation and the faults relating to each of the subsets of the timings extracted.

It is intended that a quantitative assessment will be available by the end of the project since there is now enough failure data to drive the models but the results are not available at the time of writing of this document.

References for all chapters

References

Abadi, M. and Lamport, L. (1988) *The Existence of Refinement Mappings*, DEC Research Center, Technical Report N°SRC-29.

Adams, N. (1984), "Optimizing Preventive Service of Software Products", *IBM Journal of Research and Development*, 28 (1), pp. 2-14.

Adrion, W.R., Branstad, M.A. and Cherniavsky, J.C. (1982), "Validation, Verification and Testing of Computer Software", *Computing Surveys*, 14 (2), pp. 159-192.

Aguera, M., Arlat, J., Crouzet, Y., Fabre, J.-C., Martins, E. and Powell, D. (1989) *Results of Fault-injection into an MCS Network Attachment Controller with Limited Self-Checking*, LAAS-CNRS, Report N°89.071 (Delta-4 Document N°R89.022/I2/P) (available from LAAS-CNRS, 7 Ave. Colonel Roche, 31077 Toulouse, France).

Almes, G.T., Black, A.P., Lazowska, E.D. and Noe, J.D. (1985), "The Eden System: a Technical Review", *IEEE Transactions on Software Engineering*, SE-11 (1), pp. 43-59.

Alpern, B. and Schneider, F.B. (1987), "Recognizing Safety and Liveness", *Distributed Computing*, 2, pp. 117–126.

Alur, R. and Henzinger, T.A. (1989) "A Really Temporal Logic", in *Proc. 30th. Symposium on Foundations of Computer Science (FOCS 89)*, pp. 164-169 (IEEE).

Alur, R., Courcoubetis, C. and Dill, D. (1990) "Model-Checking for Real-time Systems", in *Proc. 5th. Symposium on Logic in Computer Science (LICS 90)*, pp. 414-425 (IEEE).

Anderson, D.A. and Metze, G. (1972) "Design of Totally Self-Checking Check Circuits for M-out-of-N Codes", in *Proc. 2nd. Int. Symp. on Fault Tolerant Computing (FTCS-2)*, Newton, MA, USA, pp. 30-35 (IEEE).

Anderson, T. (1986) "A Structured Decision Mechanism for Diverse Software", in *Proc. 5th. Symp. on Reliability in Distributed Software and Data Base Systems*, Los Angeles, CA, USA, pp. 125-129.

Anderson, T.A. and Lee, P.A. (1981), *Fault Tolerance — Principles and Practice*, (Prentice-Hall).

ANSA (1987) *The ANSA Reference Manual*, Advanced Network System Architecture (Release 00.03) (available from Architecture Project Management Limited, 24 Hills Road, Cambridge CB2 1JP, UK).

ANSA (1989) *The ANSA Reference Manual*, Advanced Network System Architecture (Release 01.00) (available from Architecture Project Management Limited, 24 Hills Road, Cambridge CB2 1JP, UK).

Arlat, J., Aguera, M., Crouzet, Y., Fabre, J., Martins, E. and Powell, D. (1990), "Experimental Evaluation of the Fault Tolerance of an Atomic Multicast Protocol", *IEEE Transactions on Reliability*, 39 (4), pp. 455-467 (Special Issue on Experimental Evaluation of Computer Reliability).

Arlat, J., Aguera, M., Crouzet, Y., Fabre, J.-C., Martins, E. and Powell, D. (1989) *Dependability Testing Report LA1*, LAAS-CNRS, Report N°89.410 (Delta-4 Document N° R89.139/I1/P) (available from LAAS-CNRS, 7 Ave. Colonel Roche, 31077 Toulouse, France).

Arlat, J., Aguerra, M., Amat, L., Crouzet, Y., Fabre, J.-C., Laprie, J.-C., Martins, E. and Powell, D. (1990), "Fault-Injection for Dependability Validation — A Methodology and Some Applications", *IEEE Transactions on Software Engineering*, 16, pp. 166-182.

Arlat, J., Crouzet, Y. and Laprie, J.-C. (1989) "Fault Injection for Dependability Validation of Fault-Tolerant Computing Systems", in *Proc. 19th. Int. Symp. on Fault Tolerant Computing (FTCS-19)*, Chicago, MI, USA, pp. 348-355 (IEEE).

Arlat, J., Crouzet, Y., Martins, E. and Powell, D. (1991) *Dependability Testing Report LA2 - Fault-Injection on the Fail-Silent NAC: Preliminary Results*, LAAS-CNRS, Report N°91.043 (Delta-4 Document N° R90.206/I1/P) (available from LAAS-CNRS, 7 Ave. Colonel Roche, 31077 Toulouse, France).

Arlat, J., Kanoun, K. and Laprie, J.-C. (1988) "Dependability Evaluation of Software Fault-Tolerance", in *18th. Int. Symp. on Fault Tolerant Computing (FTCS-18)*, Tokyo, Japan, pp. 142-147 (IEEE).

Arnold, T.F. (1973), "The Concept of Coverage and its Effect on the Reliability Model of Repairable Systems", *IEEE Transactions on Computers*, C-22 (3), pp. 251-254.

Avizienis, A. (1975), "Fault-Tolerance and Fault-Intolerance: Complementary Approaches to Reliable Computing", *ACM SIGPLAN Notices*, 10 (6), pp. 458-464.

Avizienis, A. (1978), "Fault Tolerance, the Survival Attribute of Digital Systems", *Proceedings of the IEEE*, 66 (10), pp. 1109-1125.

Avizienis, A. and Kelly, J.P.J. (1984), "Fault-Tolerance by Design Diversity: Concepts and Experiments", *Computer*, 17 (8), pp. 67-80.

Avizienis, A. and Laprie, J.-C. (1986), "Dependable Computing: from Concepts to Design Diversity", *Proceedings of the IEEE*, 74 (5), pp. 629-638.

Avizienis, A., Gunningberg, P., Kelly, J.P.J., Strigini, L., Traverse, P.J., Tso, K.S. and Voges, U. (1985) "The UCLA DeDiX (Design Diversity Experiment) SYSTEM: A Distributed Testbed for Multiple-version Software", in *Proc. 15th. Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, Ann Arbor, MI, USA, pp. 126-134 (IEEE).

Babaoglu, O. and Drummond, R. (1985), "Streets of Byzantium: Network Architectures for Fast Reliable Broadcasts", *IEEE Transactions on Software Engineering*, SE-11 (6), pp. 546-554.

Babaoglu, O. and Drummond, R. (1987) "(Almost) No Cost Clock Synchronization", in *Proc. 17th. Int. Symp. on Fault-Tolerant Computing (FTCS-17)*, Pittsburgh, PA, USA, pp. 42-47 (IEEE).

Baeten, J.C.M. and Bergstra, J.A. (1990) *Real Time Process Algebra*, University of Amsterdam.

Banâtre, J.P., Banâtre, M. and Muller, G. (1988) "Main Aspects of the GOTHIC Distributed System", in *European Teleinformatics Conf. (EUTECO'88): Research into Networks and Distributed Applications*, Vienna, Austria (R. Speth, Eds.), pp. 747-760 (Elsevier Science Publishers B.V. (North-Holland)).

Banâtre, J.P., Banâtre, M., Lapalme, G. and Ployette, F. (1986), "The Design and Building of ENCHERE, a Distributed Electronic Marketing System", *Communications of the ACM*, 29 (1), pp. 19-29.

Baptista, M., Graf, S., Richier, J.-L., Rodrigues, L., Rodriguez, C., Veríssimo, P. and Voiron, J. (1990) "Formal Specification and Verification of a Network Independent Atomic Multicast Protocol", in *3rd. Int. Conf. on Formal Description Techniques (FORTE'90)* (J. Quemada, J. Mañas and E. Vazquez, Eds.) (North-Holland).

Barrett, P.A., Hilborne, A.M., Bond, P.G., Seaton, D.T., Veríssimo, P., Rodrigues, L. and Speirs, N.A. (1990) "The Delta-4 XPA Extra Performance Architecture", in *Proc. 20th. Int. Symp. on Fault-Tolerant Computing Systems (FTCS-20)*, Newcastle upon Tyne, UK, pp. 481-488 (IEEE).

Bell, D.E. and LaPadula, L.J. (1974) *Secure Computer Systems: Unified Exposition and Multics Interpretation*, The MITRE Corporation N°MTR-2997 (AD/A-020-445).

Béounes, C. and Laprie, J.-C. (1985) "Dependability Evaluation of Complex Computer Systems: Stochastic Petri Net Modeling", in *Proc. 15th. Int. Symp. Fault Tolerant Computing (FTCS-15)*, Ann Arbor, MI, USA, pp. 364-369 (IEEE).

Bergstra, J.A. and Klop, J.W. (1985), "Algebra of Communicating Processes with Abstraction", *Theoretical Computer Science*, 37, pp. 77–121.

Bernstein, P.A. (1988), "Sequoia: A Fault Tolerant Tightly Coupled Multiprocessor for Transaction Processing", *IEEE Computer*, 21 (2), pp. 37-45.

Bernstein, P.A., Hadzilacos, V. and Goodman, N. (1987), *Concurrency Control and Recovery in Database Systems*, (Addison-Wesley).

Berry, G. and Cosserat, L. (1985) "The ESTEREL Synchronous Programming Language and its Mathematical Semantics", in *Proc. CMU Seminar on Concurrency*, Lecture Notes in Computer Science, 197, pp. 389-448 (Springer-Verlag).

Beyaert, B., Florin, G., Lonc, P. and Natkin, S. (1981) "Evaluation of Computer Systems Dependability using Stochastic Petri Nets", in *Proc. 11th. Int. Symp. Fault-Tolerant Computing (FTCS-11)*, Portland, Maine, USA, pp. 79-81 (IEEE).

Birman, K. and Joseph, T. (1987), "Reliable Communication in the Presence of Failures", *ACM Transactions on Computer Systems*, 5 (1).

Birman, K.P. (1985), "Replication and Fault-Tolerance in the ISIS System", *ACM Operating Systems Review*, 19 (5), pp. 79-86 (Proc. 10th. ACM Symp. on Operating System Principles, Orcas Island, WA, USA, December 1985).

Birman, K.P. and Joseph, T.A. (1987), "Exploiting Virtual Synchrony in Distributed Systems", *ACM Operating Systems Review*, 21 (5), pp. 123-128 (Proc. 11th. ACM Symp. on Operating System Principles, Austin, TX, USA, November 1987).

Birrell, A. and Nelson, B. (1984), "Implementing Remote Procedure Calls", *ACM Transactions on Computer Systems*, 2 (1), pp. 39-59.

Bishop, P.G. (1988), "The PODS Diversity Experiment", in *Software Diversity in Computerized Control Systems*, (U. Voges, Eds.), pp. 51-84 (Springer-Verlag).

Black, A., Hitchinson, N., Jul, E., Levy, H. and Carter, L. (1987), "Distribution and Abstract Types in Emerald", *IEEE Transactions on Software Engineering*, SE-13 (1), pp. 65-75.

Boehm, B.W. (1979) "Guidelines for Verifying and Validation Software Requirements and Design Specifications", in *Proc. EURO IFIP'79*, London, UK, pp. 711-719 (IFIP).

Boehm, B.W. (1981), *Software Engineering Economics*, (Prentice Hall).

Boehm, B.W. (1988), "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, 21 (5), pp. 61-72.

Bond, P.G. (1987) *Real-Time Extensions to UNIX*, Delta-4 Project Consortium, Delta-4 Document (Specification S5, Phase 1).

Bond, P.G., Drackley, S.D., Powell, D. and Seaton, D.T. (1987) *The Impact of Real-Time and Dependability on Open, Distributed Systems - Interim Analysis of Requirements: Input to ECMA TC32/TG2*, Delta-4 Project Consortium, LAAS Report N°87.415 (available from LAAS-CNRS, 7 Ave. Colonel Roche, 31077 Toulouse, France).

Borg, A., Baumbach, J. and Glazer, S. (1983) "A Message System supporting Fault Tolerance", in *Proc. 9th. Symp. on Operating System Principles*, pp. 90-99 (ACM).

Bouajjani, A., Fernandez, J.-C. and Halbwachs, N. (1990) "Minimal Model Generation", in *Proc. Conf. on Automatic Verification (CAV 90)*, Rutgers, NJ, USA, Lecture Notes on Computer Science, 531 (Springer Verlag).

Bouajjani, A., Fernandez, J.-C., Graf, S., Rodriguez, C. and Sifakis, J. (1991) "Safety for Branching Semantics", in *Proc. 18th. Int. Conf. on Automata, Languages and Programming (ICALP 91)*, Lecture Notes in Computer Science, 510, pp. 72-92 (Springer-Verlag).

Bouricius, W.G., Carter, W.C. and Schneider, P.R. (1969) "Reliability Modeling Techniques for Self-Repairing Computer Systems", in *Proc. 24th. National Conference*, pp. 295-309 (ACM).

Boyer, R.S. and Moore, J.S. (1979), *A Computational Logic*, (Academic Press).

Brookes, S.D. and Rounds, S.D. (1983) "Behavioral Equivalence Relations induced by Programming Logics", in *Proc. 10th. Int. Conf. on Automata, Languages and Programming (ICALP 83)*, Lecture Notes in Computer Science, 154, pp. 97-108 (Springer Verlag).

Brookes, S.D., Hoare, C.A.R. and Roscoe, A.W. (1984), "A Theory of Communicating Sequential Processes", *Journal of the ACM*, 31 (3), pp. 560–599.

Brooks, F.P. (1987), "No Silver Bullett - Essence and Accidents of Software Engineering", *IEEE Computer*, 20 (4), pp. 10-19.

Büchi, J.R. (1962) "On a Decision Method in Restricted Second Order Arithmetic", in *Proc. Int. Congress on Logic, Method and Philosophy of Science*, pp. 1-11 (Stanford University Press).

Burns, A. (1990a) "Distributed Hard Real-time Systems: what Restrictions are Necessary?", in *Proc. 1989 Real-Time Symp.* (H. Zedan, Eds.) (Elsevier Scientific).

Burns, A. (1990b) *Scheduling Hard Real-Time Systems: A Review*, Department of Computing, University of Bradford, UK, Research Report.

Cart, M., Ferrie, J. and Mardyanto, S. (1987) "Atomic Broadcast Protocol, Preserving Concurrency for an Unreliable Broadcast Network", in *Proc. IFIP Conf. on Local Communication Systems: LAN and PBX* (J. Cabanel, G. Pujole and A. Danthine, Eds.) (North-Holland).

Carter, W.C. (1979) "Fault Detection and Recovery Algorithms for Fault-Tolerant Systems", in *Proc. EURO IFIP'79*, London, UK, pp. 725-734 (IFIP).

Carter, W.C. (1982) "A Time for Reflection", in *Proc. 12th. Int. Symp. on Fault Tolerant Computing (FTCS-12)*, Santa Monica, CA, USA, pp. 41 (IEEE).

Carter, W.C. and Schneider, P.R. (1968) "Design of Dynamically Checked Computers", in *Proc. IFIP'68 Cong.*, Amsterdam, The Netherlands, pp. 878-883 (IFIP).

Carter, W.C., Joyner, W.H., Brand, D., Ellozy, H.A. and Wolf, J.L. (1978) "An Improved System to Verify Assembled Programs", in *Proc. 8th. Int. Symp. on Fault Tolerant Computing (FTCS-8)*, Toulouse, France, pp. 165-170 (IEEE).

Caspi, P., Halbwachs, N., Pilaud, N. and Plaice, J. (1987) "LUSTRE, a Declarative Language for Programming Synchronous Systems", in *Proc. 14th. Symp. on Principles of Programming Languages (POPL 87)*, Munich, Germany, pp. 178-188 (ACM).

Castillo, X. and Siewiorek, D.P. (1981) "Workload, Performance and Reliability of Digital Computing Systems", in *Proc. 11th. Int. Symp. on Fault Tolerant Computing (FTCS-11)*, Portland, Maine, USA, pp. 84-89 (IEEE).

Chang, J. and Maxemchuck, N. (1984), "Reliable Broadcast Protocols", *ACM Transactions on Computer Systems*, 2 (3).

Cheheyl, M.H., Gasser, M., Huff, G.A. and Miller, J.K. (1981), "Verifying Security", *ACM Computing Surveys*, 13 (3), pp. 279-339.

Chen, L. and Avizienis, A. (1978) "N-Version-Programming: A Fault-Tolerance Approach to Reliability of Software Operation", in *Proc. 8th. Int. Symp. on Fault-Tolerant Computing (FTCS-8)*, Toulouse, France, pp. 3-9 (IEEE).

Cheriton, D. and Zwaenepoel, W. (1985), "Distributed Process Groups in the V-Kernel", *ACM Transactions on Computer Systems*, 3 (2).

Chesson, G. (1988) "XTP/PE Overview", in *Proc. 13th. Local Computer Network Conference*, Minneapolis-USA (IEEE).

Chillarege, R. and Bowen, N.S. (1989) "Understanding Large System Failures — A Fault Injection Experiment", in *Proc. 19th. Int. Symp. on Fault-Tolerant Computing (FTCS-19)*, Chicago, MI, USA, pp. 356-363 (IEEE).

Choi, G.S., Iyer, R.K. and Carreno, V. (1989) "A Fault Behavior Model for an Avionic Microprocessor: a Case Study", in *Proc. 1st Int. Working Conf. on Dependable Computing for Critical Applications*, Santa Barbara, CA, USA (A. Avizienis and J.-C. Laprie, Eds.), pp. 171-195 (Springer-Verlag).

Ciompi, P. and Grandoni, F. (1990) *SWFT in Delta-4: Selection of Techniques*, Delta-4 Project Consortium, Delta-4 Document N°R89.146/I1/P (available from Ferranti International, Wythenshawe, Manchester M22 5LA, UK).

Clark, D.D. and Wilson, D.R. (1987) "A Comparison of Commercial and Military Computer Security Policies", in *Proc. Symp. on Security and Privacy*, Oakland, CA, USA, pp. 184-194 (IEEE).

Clarke, E.M., Emerson, E.A. and Sistla, E. (1986), "Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications: A Practical Approach", *ACM Transactions on Programming Languages and Systems*, 8 (2), pp. 244-263.

Cleaveland, R., Parrow, J.G. and Steffen, B. (1989) "The Concurrency Workbench", in *Proc. Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, Lecture Notes in Computer Science, 407, pp. 24-37 (Springer Verlag).

CNMA (1990) *Network Management*, ESPRIT Project 2617: Communications Network for Manufacturing Applications (CNMA).

Cooper, E.C. (1984) "Circus: A Replicated Procedure Call Facility", in *Proc. 4th. Symp. on Reliability in Distributed Software and Database Systems*, Maryland, USA, pp. 11-24 (Silver Spring).

Côrtes, M.L., Millman, S.D., Goosen, H.A. and McCluskey, E.J. (1987) *Techniques for Injecting Non Stuck-at Faults*, Center for Reliable Computing (CRC), Technical Report N°87-21.

Costes, A., Doucet, J.-E., Landrault, C. and Laprie, J.-C. (1981) "SURF: a Program for Dependability Evaluation of Complex Fault-Tolerant Computing Systems", in *Proc. 11th. Int. Symp. Fault-Tolerant Computing (FTCS-11)*, Portland, Maine, USA, pp. 72-78 (IEEE).

Cox, D.R. and Miller, H.D. (1968), *The Theory of Stochastic Processes*, (Methuen).

Craigen, D. (1987) "Strengths and Weaknesses of Program Verification Systems", in *Proc. 1st European Software Engineering Conf.*, Strasbourg, France, pp. 421-429.

Cristian, F. (1980) "Exception Handling and Software Fault Tolerance", in *Proc. 10th. Int. Symp. on Fault Tolerant Computing (FTCS-10)*, Kyoto, Japan, pp. 97-103 (IEEE) (also in *IEEE Transactions on Computers*, C-31(6), pp. 531-540, June 1982).

Cristian, F. (1987) *Exception Handling*, IBM Almaden Research Center, RJ5724 (57703).

Cristian, F. (1988) "Agreeing on Who is Present and Who is Absent in a Synchronous Distributed System", in *Proc. 18th. Int. Symp. on Fault-Tolerant Computing (FTCS-18)*, Tokyo, Japan, pp. 206-211 (IEEE).

Cristian, F. (1989), "Probabilistic Clock Synchronization", *Distributed Computing*, 3 (3), pp. 146-158.

Cristian, F. (1991), "Understanding Fault-Tolerant Distributed Systems", *Communications of the ACM*, 34 (2), pp. 56-78.

Cristian, F., Aghali, H., Strong, R. and Dolev, D. (1985) "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement", in *Proc. 15th. Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, Ann Arbor, MI, USA, pp. 200-206 (IEEE).

Cristian, F., Dancey, B. and Dehn, J. (1990) "Fault-Tolerance in the Advanced Automation System", in *Proc. 20th. Int. Symp. on Fault-Tolerant Computing (FTCS-20)*, Newcastle upon Tyne, UK, pp. 6-17 (IEEE).

Cristian, F., H., A. and Strong, R. (1986) "Clock Synchronization in the Presence of Omission and Performance Faults", in *Proc.16th. Int. Symp. on Fault-Tolerant Computing (FTCS-16)*, Vienna, Austria (IEEE).

Crouzet, Y. and Decouty, B. (1982) "Measurements of Fault Detection Mechanisms Efficiency: Results", in *Proc. 12th. Int. Symp. Fault-Tolerant Computing (FTCS-12)*, Santa Monica, CA, USA, pp. 373-376 (IEEE).

Damm, A. (1988) *Experimental Evaluation of Error-detection and Self-Checking Coverage of Components of a Distributed Real-time System*, Doctoral Dissertation, Technical University, Vienna, Austria.

David, R. (1986), "Signature Analysis for Multiple Output Circuits", *IEEE Transactions on Computers*, C-35 (9), pp. 830-837.

David, R. and Thévenod-Fosse, P. (1981), "Random Testing of Integrated Circuits", *IEEE Transactions on Instrumentation and Measurement*, IM-30 (1), pp. 20-25.

Davies, J. and Schneider, S. (1989) *An Introduction to Timed CSP*, Oxford University Computing Laboratory, Technical Monograph N°PRG-75.

Delta-4 (1990) *Delta-4 Application Support Environment*, Implementation Guide IG2, Delta-4 Project Consortium, Delta-4 Document (available from Bull SA, 1 Rue de Provence, 38423 Echirolles, France).

Delta-4 (1991) *Implementation Guide IG2*, Delta-4 Project Consortium, Delta-4 Document (available from Bull SA, 1 Rue de Provence, 38423 Echirolles, France).

DeMillo, R.A., Lipton, R.J. and Sayward, F.G. (1978), "Hints on Test Data Selection: Help for the Practicing Programmer", *Computer*, 11 (4), pp. 34-41.

Denning, D.E. (1982), *Cryptography and Data Security*, (Addison-Wesley).

Denning, D.E. (1986) "An Intrusion-Detection model", in *Proc. Symp. on Security and Privacy*, Oakland, CA, USA, pp. 118-132 (IEEE).

Dertouzos, M.L. and Mok, A.K. (1989), "Multiprocessor On-Line Scheduling of Hard Real-Time Tasks", *IEEE Transaction on Software Engineering*, 15 (12), pp. 1497-1506.

Deswarte, Y., Blain, L. and Fabre, J.-C. (1991) "Intrusion Tolerance in Distributed Systems", in *Proc. Symp. on Research in Security and Privacy*, Oakland, CA, USA, pp. 110-121 (IEEE).

Di Giandomenico, F. and Strigini, L. (1990) "Adjudicators for Diverse-redundant Components", in *Proc. 9th. Symp. on Reliable Distributed Systems (SRDS-9)*, Huntsville, AL, USA.

Diaz, M. (1982), "Modeling and Analysis of Communication and Cooperation Protocols using Petri Net Based Models", *Computer Networks*, 6 (6), pp. 419-441.

DoD 5200.28 (1985) *Trusted Computer System Evaluation Criteria*, US Department of Defense, STD N°5200.28.

Duane, J.T. (1964), "Learning Curve Approach to Reliability Monitoring", *IEEE Transactions on Aerospace*, 2, pp. 563-566.

Dugan, J.B. and Trivedi, K.S. (1989), "Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems", *IEEE Transactions on Computers*, 38 (6), pp. 775-787.

Duran, J.W. and Ntafos, S.C. (1984), "An Evaluation of Random Testing", *IEEE Transactions on Software Engineering*, SE-10 (4), pp. 438-444.

Eckhard, D.E. and Lee, L.D. (1985), "A Theoretical Basis for the Analysis of Multiversion Software subject to Coincident Errors", *IEEE Transactions on Software Engineering*, SE-11 (12), pp. 1511-1517.

ECMA 127 (1990) *Remote Procedure Call using OSI (RPC)*, ECMA N°127.

ECMA TR/49 (1990) *Support Environment for Open Distributed Processing (SE-ODP)*, ECMA, Technical Report N°49.

Eich, M.H. (1988), "Graph Directed Locking", *IEEE Transactions on Software Engineering*, 14 (2), pp. 133-140.

Elmendorf, W.R. (1972) "Fault-Tolerant Programming", in *Proc. 2nd. Int. Symp. on Fault Tolerant Computing (FTCS-2)*, Newton, MA, USA, pp. 79-83 (IEEE).

Emerson, E.A. and Halpern, J.Y. (1986), "'Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time Temporal Logic", *Journal of the ACM*, 33 (1), pp. 151-178.

Emerson, E.A., Mok, A.K., Sistla, A.P. and Srinivasan, J. (1989) "Quantitative Temporal Reasoning", in *Proc. Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, pp. 15 (Participants' version of Proceedings).

Ezhilchelvan, P.D. and Shrivastava, S.K. (1986) "A Characterization of Faults in Systems", in *Proc. 5th. Symp. on Reliability in Distributed Software and Database Systems*, Los Angeles, CA, USA, pp. 215-222 (IEEE).

Ezhilchelvan, P.D. and Shrivastava, S.K. (1991) "A Distributed Systems Architecture supporting Availability and Reliability", in *Proc. 2nd. Conf. on Dependable Computing for Critical Applications*, Tucson, AZ, USA (IFIP).

Fabry, R.S. (1974), "Capability-Based Addressing", *Communications of the ACM*, 17 (7), pp. 403-412.

Fernandez, J.-C. (1990), "An Implementation of an Efficient Algorithm for Bisimulation Equivalences", *Science of Computer Programming*, 13 (2-3), pp. 219-236.

Fernandez, J.-C. and Mounier, L. (1990) "Verification Bisimulations on the Fly", in *Proc. 3th. Int. Conf. on Formal Description Techniques (FORTE'90)* (J. Quemada, J. Mañas and E. Vazquez, Eds.) (North-Holland).

Fernandez, J.-C., Richier, J.-L. and Voiron, J. (1985) "Verification of Protocol Specifications using the Xesar System", in *Proc IFIP WG6.1 5th. International Conference on Protocol, Specification, Testing and Verification*, pp. 71-90 (North-Holland).

Fischer, M.J. (1983) "The Consensus Problem in Unreliable Distributed Systems (A Brief Survey)", in *Proc. Int. Conf. on Foundations of Computations Theory*, Borgholm, Sweden, pp. 127-140.

Fischer, M.J., Lynch, N.A. and Paterson, M.S. (1985), "Impossibility of Distributed Consensus with One Faulty Process", *Journal of the ACM*, 32 (2), pp. 374-382.

Floyd, R.W. (1967) "Assigning Meaning to Programs", in *Proc. Symposium in Applied Maths*, Providence, RI, USA, vol. XIX , pp. 19–32 (AMS).

Fraga, J. (1985) *Data Security by Intrusion-Tolerance*, Doctoral Dissertation, Institut National Polytechnique, Toulouse, France (LAAS Report N°85.133, in French).

Fraga, J. and Powell, D. (1985) "A Fault and Intrusion-Tolerant File System", in *Proc. 3rd. IFIP Int. Cong. on Computer Security*, Dublin, Ireland, pp. 203-218 (North-Holland).

Francez, N. (1986), *Fairness*, Monographs in Computer Science, (Springer Verlag).

Fray, J.M., Deswarte, Y. and Powell, D. (1986) "Intrusion-Tolerance using Fine-Grain Fragmentation-Scattering", in *Proc. Symp. on Security and Privacy*, Oakland, CA, USA, pp. 194-201 (IEEE).

Frison, S.G. and Wensley, J.H. (1982) "Interactive Consistency and its Impact on the Design of TMR Systems", in *Proc. 12th. Int. Symp. on Fault Tolerant Computing (FTCS-12)*, Santa Monica, CA, USA, pp. 228-233 (IEEE).

FTCS12 (1982) *Fundamental Concepts of Fault-Tolerance*, Proc. 12th. Int. Symp. on Fault-Tolerant Computing (FTCS-12), IEEE.

Garcia-Molina, H., Kogan, B. and Lynch, N. (1988) "Reliable Broadcast in Networks with Nonprogrammable Servers", in *Proc. 8th. Int. Conf. on Distributed Computing Systems (ICDCS-8)*, San Jose, CA, USA, pp. 428-437 (IEEE).

Gasser, M. (1988), *Building a Secure Computer System*, (Van Nostran Reinhold).

Gérardin, J.P. (1986), "Design Aid to Reliable and Safe Systems: The DEFI", *Electronique Industrielle* 116, pp. 58-63 (in French).

Giloth, F.K. and Prantzen, K.D. (1983) "Can the Reliability of Digital Telecommunication Switching Systems be Predicted and Measured?", in *Proc. 13th. Int. Symp. on Fault-Tolerant Computing (FTCS-13)*, Milano, Italy, pp. 392-397 (IEEE).

Goel, A.L. and Okumoto, K. (1979), "Time-Dependent Error-Detection Rate Model for Software and other Performance Measures", *IEEE Transactions on Reliability*, R-28 (3), pp. 465-484.

Golberg, A. and Robson, D. (1981), *Smalltalk-80: The Language and its Implementation*, (Addison-Wesley).

Goldberg, J. (1982) "A Time for Integration", in *Proc. 12th. Int. Symp. on Fault Tolerant Computing (FTCS-12)*, Santa Monica, CA, USA, pp. 42 (IEEE).

Goodenough, J.B. and Gerhart, S.L. (1975), "Toward a Theory of Test Data Selection", *IEEE Transactions on Software Engineering*, SE-1 (2), pp. 156-173.

Gordon, M., Milner, R. and Wadsworth, C. (1979), *Edinburgh LCF*, (Springer Verlag).

Graf, S. and Sifakis, J. (1986), "A Logic for the Description of Non-Deterministic Programs and their Properties", *Information and Control*, 68 (1-3), pp. 125-145.

Graf, S. and Steffen, B. (1990) "Compositional Minimization of Finite State Systems", in *Proc. Conf. on Automatic Verification (CAV 90)*, Rutgers, NJ, USA, Lecture Notes in Computer Science, 531 (Springer Verlag).

Graf, S., Richier, J.-L., Rodriguez, C. and Voiron, J. (1989) *Protocol Verification Report*, Delta-4 Project Consortium, Delta-4 Document N°R89.147/I1/P (available from Ferranti International, Wythenshawe, Manchester M22 5LA, UK).

Graf, S., Richier, J.-L., Rodriguez, C. and Voiron, J. (1990) *Protocol Verification Report*, Delta-4 Project Consortium, Delta-4 Document N°R90.225 (available from Ferranti International, Wythenshawe, Manchester M22 5LA, UK).

Graf, S., Richier, J.L., Rodriguez, C. and Voiron, J. (1989) "What are the Limits of Model Checking for the Verification of Real Life Protocols", in *Proc. Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, Lecture Notes on Computer Science, 407, pp. 275-285 (Springer-Verlag).

Gray, J. (1986) "Why do Computers Stop and What can be done about it?", in *Proc. 5th. Symp. on Reliability in Distributed Software and Database Systems*, Los Angeles, CA, USA, pp. 3-12 (IEEE).

Groote, G.F. (1990) *Specification and Verification of Real Systems in ACP*, CWI, Amsterdam, Technical Report N°CS-R9015.

Guérin, C., Raison, H. and Martin, P. (1985) *Procedure for Dependable Message Broadcasting in a Ring and Mechanism for Implementing the Procedure*, French Patent N°85.2.2 (in French).

Gunneflo, U., Karlsson, J. and Torin, J. (1989) "Evaluation of Error Detection Schemes using Fault Injection by Heavy-ion Radiation", in *Proc. 19th. Int. Symp. Fault-Tolerant Computing (FTCS-19)*, Chicago, MI, USA, pp. 340-347 (IEEE).

Gunningberg, P. (1983) "Voting and Redundancy Management Implemented by Protocols in Distributed Systems", in *Proc. 13th. Int. Symp. on Fault-Tolerance Computing (FTCS-13)*, Milano, Italy, pp. 182-185 (IEEE).

Habermann, A.N. (1969), "Prevention of System Deadlocks", *Communications of the ACM*, 12 (7), pp. 373-385.

Halang, W.A. (1986), "Implications on Suitable Multiplrocessor Structures and Virtual Storage Management when applying a Feasible Scheduling Algorithm in Hard Real-time Environments", *Software Practice and Experience*, 16 (8), pp. 761-769.

Halpern, J.Y., Simons, B., Strong, H.R. and Dolev, D. (1984) "Fault-Tolerant Clock Synchronisation", in *Proc. 3rd. Symp. on the Principles of Distributed Computing*, Vancouver, Canada, pp. 89-102 (ACM).

Hamlet, R. (1987) "Testing for Trustworthiness", in *Proc. DIAC 87 Symp. on Directions and Implications of Advanced Computing*, Seattle, WA, USA, pp. 87-93.

Har'El, Z. and Kurshan, R.P. (1990), "Software for Analytical Development of Communication Protocols", *AT&T Technical Journal*, 60 (1), pp. 45-58.

Harel, D., Pnueli, A., Schmidt, J.P. and Sherman, R. (1987) "On the Formal Semantics of State Charts", in *Proc. 2nd Symp. on Logic in Computer Science (LICS 87)*, pp. 54-64 (IEEE).

Harel, E., Lichtenstein, O. and Pnueli, A. (1990) "Explicit Clock Temporal Logic", in *Proc. 5th. Symp. on Logic in Computer Science (LICS 90)*, pp. 402-413 (IEEE).

Haugk, G., Lax, F.M., Rover, R.D. and Williams, J.R. (1985), "The 5 ESS Switching System: Maintenance Capabilities", *AT&T Technical Journal*, 64 (6), pp. 1385-1416.

Hecht, H. (1976), "Fault-Tolerant Software for Real-time Applications", *ACM Computing Surveys*, 8 (4), pp. 391-407.

Hecht, H. and Fiorentino, E. (1987) "Reliability Assessment of Spacecraft Electronics", in *Proc. Annual Reliability and Maintainability Symp.*

Hennessy, M. and Milner, R. (1985), "Algebraic Laws for Nondeterminism and Concurrency", *Journal of the ACM*, 32 (1), pp. 137-161.

Hennessy, M. and Regan, T. (1990) *A Temporal Process Algebra*, University of Sussex, Computer Science, Report N°2/90.

Hetzel, W. (1984), *The Complete Guide to Software Testing*, (QED Information Science).

Hoare, C.A.R. (1969), "An Axiomatic Basis for Computer Programming", *Communications of the ACM*, 12 (10), pp. 576-583.

Hoare, C.A.R. (1972), "Proof of Correctness of Data Representations", *Acta Informatica*, 1, pp. 271–281.

Hoare, C.A.R. (1985), *Communicating Sequential Processes*, (Prentice Hall International).

Holzmann, G.J. (1984), "The Pandora System: an Interactive System for the Design of Data Communication Protocols", *Computer Networks*, 8 (2), pp. 71–81.

Holzmann, G.J. (1990), "Algorithms for Automated Protocol Validation", *AT&T Technical Journal*, 60 (1), pp. 32–44.

Horning, J.J., Lauer, H.C., Melliar-Smith, P.M. and Randell, B. (1974), "A Program Structure for Error Detection and Recovery", in *Operating Systems*, (G. Goos and J. Hartmanis, Eds.), Lectures Notes in Computer Science, 16, pp. 172-187 (Springer-Verlag).

Hosford, J.E. (1960), "Measures of Dependability", *Operations Research*, 8 (1), pp. 204-206.

Howard, R.A. (1971), *Dynamic Probabilistic Systems*, (John Wiley).

Howden, W.E. (1987), *Functional Program Testing and Analysis*, (McGraw-Hill).

Huang, K.K. and Abraham, J.A. (1982) "Low Cost Schemes for Fault Tolerance in Matrix Operations with Processor Arrays", in *Proc. 12th. Int. Symp. on Fault Tolerant Computing (FTCS-12)*, Santa Monica, CA, USA, pp. 330-337 (IEEE).

IEC 191 (1985) *Reliability, Maintainability and Quality of Service*, International Electrotechnical Vocabulary, Chapter 191, International Electrotechnical Commission, Document N°1-IEV-191-Central Office-1243 and 56-IEV-191-Central Office-119 (Geneva).

IEEE 729 (1982) *Standard Glossary of Software Engineering Terminology*, IEEE N°729.

IEEE 829 (1983) *Standard for Software Test Documentation*, IEEE N°829.

IEEE P1003 (1989) *Realtime Extension for Portable Operating Systems*, IEEE N°P1003/P9.

ISO 7498-1 *Basic Reference Model*, Information Processing Systems: Open Systems Interconnection, IS N°7489-1 (1984).

ISO 7498-4 *Basic Reference Model, Part 4: OSI Management Framework*, Information Processing Systems: Open Systems Interconnection, ISO, IS N°7498-4 (E) (1989).

ISO 8571 *File Transfer and Access Method (FTAM)*, Information Processing Systems: Open Systems Interconnection, ISO, IS N°8571.

ISO 8802-4 *Token-passing Bus Access Method and Physical Layer Specifications*, ISO, DIS N°8802-4 (1984).

ISO 8802-5 *Token Ring Access Method and Physical Layer Specifications*, ISO, DP N°8802-5 (1985).

ISO 8807 *LOTOS: a Formal Description Technique based on the Temporal Ordering of Observational Behaviour*, ISO, IS N°8807 (1989).

ISO 9074 *ESTELLE: A Formal Description Technique based on an Extended State Transition Model*, ISO, IS N°9074 (1989).

ISO 9314 *Fiber Distributed Data Interface (FDDI)*, ISO, DP N°9314 (ANSI Standard X3.139, 1987).

ISO 9506 *Manufacturing Message Specification (MMS)*, Information Processing Systems: Open Systems Interconnection, ISO, IS N°9506.

ISO 9595 *Common Management Information Service Definition*, Information Processing Systems: Open Systems Interconnection, ISO/IEC N°9595 (E) (1991).

ISO 10040 *Systems Management Overview*, Information Processing Systems: Open Systems Interconnection, ISO, DIS N°10040 (E) (1990).

ISO 10165-1 *Management Information Services - Structure of Management Information, Part 1: Management Information Model*, Information Processing Systems: Open Systems Interconnection, ISO/IEC, DIS N°10165-1 (E) (1990).

ISO 10165-4 *Management Information Services - Structure of Management Information, Part 4: Guidelines for the Definition of Managed Objects (GDMO)*, Information Processing Systems: Open Systems Interconnection, ISO/IEC, DIS N°10165-4 (E) (1990).

ISO N2031 *Working Draft addendum to ISO 7498-1 on Multipeer Data Transmission*, ISO N°TC97/SC21 WG1 N2031 (1987).

Iyer, R.K., Butner, S.E. and McCluskey, E.J. (1982), "A Statistical Failure/Load Relationship: Results of a Multi-Computer Study", *IEEE Transactions on Computers*, C-31, pp. 697-706.

Jard, C. and Jeron, T. (1989) "On-Line Model Checking for Finite Linear Temporal Logic Specifications", in *Proc. Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, Lecture Notes on Computer Science, 407, pp. 189-196 (Springer Verlag).

Jensen, E.D. (1991) *Alpha: a Non-Proprietary Experimental Operating System for Distributed Mission-Critical Real-Time Applications — An Overview of its Objectives and Kernel Abstractions*, Concurrent Computer Corporation, Draft (available from Concurrent Computer Corporation, One Technology Way, Westford, MA 01886, USA).

Jensen, E.D. and Northcutt, J.D. (1990) "Alpha: a Non-Proprietary Operating System for Mission-Critical Real-Time Distributed Systems", in *Proc. Workshop on Experimental Distributed Systems*, Huntsville, AL, USA (IEEE).

Jensen, E.D., Locke, C.D. and Tokuda, H. (1985) "A Time-Value Driven Scheduling Model for Real-Time Operating Systems", in *Proc. Symp. on Real-Time Systems* (IEEE).

Jessep, D.C. (1977) *Fault-Tolerant Computing, Definition of Terms*, IEEE Computer Society N°P 610/DI.

Johnson, H.H. and Madison, M.S. (1974) "Deadline Scheduling for a Real-Time Multiprocessor", in *Proc. Eurocomp Conference.*

Joseph, M.K. and Avizienis, A. (1988) "A Fault Tolerance Approach to Computer Viruses", in *Proc. 1988 Symp. on Security and Privacy*, Oakland, CA, USA, pp. 52-58 (IEEE).

Josko, B. (1987) "MCTL - An Extension of CTL for Modular Verification of Concurrent Systems", in *Proc. Workshop on Temporal Logic in Specification*, Manchester, UK, Lecture Notes on Computer Science, 398, pp. 165-187 (Springer Verlag).

Kanoun, K. and Powell, D. (1991) "Dependability Evaluation of Bus and Ring Communication Topologies for the Delta-4 Distributed Fault-Tolerant Architecture", in *Proc. 10th. Symp. on Reliable Distributed Systems (SRDS-10)*, Pisa, Italy (IEEE).

Kanoun, K. and Sabourin, T. (1987) "Software Dependability of a Telephone Switching System", in *Proc. 17th. Int. Symp. on Fault Tolerant Computing (FTCS-17)*, Pittsburgh, PA, USA, pp. 236-241 (IEEE).

Kanoun, K., Bastos, M.R. and Moreira de Souza, J. (1988) *A Method for Software Reliability Analysis and Prediction: Application to the TROPICO-R Switching System*, LAAS-CNRS, Report N°88.337.

Kieckhafer, R.M., Walter, C.J., Finn, A.M. and Thambidurai, P.M. (1988), "The MAFT Architecture for Distributed Fault Tolerance", *IEEE Transactions on Computers*, 37 (4), pp. 398-405.

Kim, K.H. (1984) "Distributed Execution of Recovery Blocks: an Approach to Uniform Treatment of Hardware and Software Faults", in *Proc. 4th. Int. Conf. on Distributed Computing Systems*, pp. 526-532.

Kim, K.H. and Ramamoorthy, C.V. (1976) "Failure-Tolerant Parallel Programming and its Supporting System Architecture", in *Proc. 4th. Int. Conf. Distributed Computing System (ICDCS-4)*, San Francisco, CA, USA, pp. 413-423 (IEEE).

Knuth, D.E. (1973), *The Art of Computer Programming*, (Addison-Wesley).

Koga, Y., Fukushima, E. and Yoshihara, K. (1982) "Error Recoverable and Securable Data Communication for Computer Network", in *Proc. 12th. Int. Symp. on Fault-Tolerant Computing (FTCS-12)*, Santa Monica, CA, USA, pp. 183-186 (IEEE).

Kopetz, H. (1974) "Software Redundancy in Real-time Systems", in *Proc. IFIP Conf. Information Processing*, Stockholm, Sweden, pp. 182-186 (North-Holland).

Kopetz, H. (1986) "Scheduling in Distributed Real-Time Systems", in *Proc. Advanced Seminar on R/T Lans*, Bandol, France (INRIA).

Kopetz, H. and Merker, W. (1985) "The Architecture of MARS", in *Proc. 15th. Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, Ann Arbor, MI, USA, pp. 274-279 (IEEE).

Kopetz, H. and Ochsenreiter, W. (1987), "Clock Synchronization in Distributed Real-Time Systems", *IEEE Transactions on Computers*, C-36 (8), pp. 933-940.

Kopetz, H. and Schwabl, W. (1989) *Global Time in Distributed Real-Time Systems*, Technische Universität Wien, Research Report N°15/89.

Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C. and Zainlinger, R. (1988), "Distributed Fault-Tolerant Real-Time Systems: The MARS Approach", *IEEE Micro*, 9 (1), pp. 25-40.

Kopetz, H., Kantz, H., Grünsteidl, G., Puscher, P. and Reisinger, J. (1990) "Tolerating Transient Faults in MARS", in *Proc. 20th. Int. Symp. on Fault-Tolerant Computing (FTCS-20)*, Newcastle upon Tyne, UK, pp. 466-473 (IEEE).

Kozen, D. (1982) "Results on the Propositional μ-calculus", in *Proc. 9th. Int. Conf. on Automata, Languages and Programming (ICALP 82)*, Lecture Notes on Computer Science, 140, pp. 348-359 (Springer Verlag).

Kuipers, B. (1985), "Commonsense Reasoning about Causality: Deriving Behavior from Structure", in *Qualitative Reasoning about Physical Systems*, (D. G. Bobrow, Eds.), pp. 169-203 (MIT Press).

Lala, J.H. (1983) "Fault Detection, Isolation and Reconfiguration in FTMP: Methods and Experimental Results", in *Proc. Digitals Avionics Systems Conf.*, pp. 21.3.1-21.3.9 (AIAA/IEEE).

Lala, J.H. (1986) "A Byzantine Resilient Fault-Tolerant Computer for Nuclear Power Plant Applications", in *Proc. 16th. Int. Symp. on Fault Tolerant Computing (FTCS-16)*, Vienna, Austria, pp. 338-343 (IEEE).

Lamport, L. (1977), "Proving the Correctness of Multiprocess Programs", *IEEE Transactions on Software Engineering*, SE-3 (2), pp. 125–143.

Lamport, L. (1978), "Time, Clocks and the Ordering of Events in a Distributed System", *Communication of the ACM*, 21 (7), pp. 558-565.

Lamport, L. (1981), "Password Authentication with Insecure Communications", *Communications of the ACM*, 24 (11), pp. 770-772.

Lamport, L. (1984), "Using Time instead of Timeout for Fault-Tolerant Distributed Systems", *ACM Transactions on Programming Languages and Systems*, 6 (2), pp. 254-280.

Lamport, L. and Melliar-Smith, P.M. (1985), "Synchronizing Clocks in the Presence of Faults", *Journal of the ACM*, 32 (1), pp. 52-78.

Lamport, L. and Schneider, F.B. (1985), "Formal Foundation for Specification and Verification", in *Distributed Systems, Methods and Tools for Specification*, Lecture Notes in Computer Science, 5 (Springer-Verlag).

Lamport, L., Shostak, R. and Pease, M. (1982), "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems*, 4 (3), pp. 382-401.

Lampson, B.W. (1981), "Atomic Transactions", in *Distributed Systems — Architecture and Implementation*, Lecture Notes in Computer Science, 11 (Springer-Verlag).

Laprie, J.-C. (1975) "Reliability and Availability of Repairable Systems", in *Proc. 5th. Int. Symp. on Fault Tolerant Computing (FTCS-5)*, Paris, France, pp. 87-92 (IEEE).

Laprie, J.-C. (1983) "A Trustable Evaluation of Computer Systems Dependability", in *Proc. Int. Workshop on Applied Mathematics and Performance/Reliability Models of Computer/Communication Systems*, Pisa, Italy (IEEE).

Laprie, J.-C. (1984), "Dependability Evaluation of Software Systems in Operation", *IEEE Transactions on Software Engineering*, SE-10 (6), pp. 701-714.

Laprie, J.-C. (1985) "Dependable Computing and Fault Tolerance: Concepts and Terminology", in *Proc. 15th. Int. Symp. on Fault Tolerant Computing (FTCS-15)*, Ann Arbor, MI, USA, pp. 2-11 (IEEE).

Laprie, J.-C. (1989), "Dependability: A Unifying Concept for Reliable Computing and Fault Tolerance", in *Dependability of Resilient Systems*, (T. Anderson, Eds.), pp. 1-28 (Blackwell Scientific Publications).

Laprie, J.-C., Arlat, J., Béounes, C., Kanoun, K. and Hourtolle, C. (1987) "Hardware and Software Fault-Tolerance: Definition and Analysis of Architectural Solutions", in *Proc. 17th. Int. Symp. on Fault-Tolerant Computing (FTCS-17)*, Pittsburgh, PA, USA, pp. 116-121 (IEEE).

Laprie, J.-C., Béounes, C., Kaâniche, M. and Kanoun, K. (1990) "The Transformation Approach to the Modeling and Evaluation of the Reliability and Availability Growth of Systems in Operation", in *Proc. 20th. Int. Symp. on Fault Tolerant Computing (FTCS-20)*, Newcastle upon Tyne, UK, pp. 364-371 (IEEE).

Laprie, J.-C., Costes, A. and Landrault, C. (1981), "Parametric Analysis of 2-unit Redundant Computer Systems with Corrective and Preventive Maintenance", *IEEE Transactions on Reliability*, R-30 (2), pp. 139-144.

Le Lann, G. (1989) "Critical Issues in Distributed Real-time Computing", in *Proc. of Workshop on Comunication Networks and Distributed Operating Systems within the Space Environment* (ESTEC).

Lehmann, D., Pnueli, A. and Stavi, J. (1981) "Impartiality, Justice and Fairness: the Ethics of Concurrent Termination", in *Proc. 8th. Int. Conf. on Automata, Languages and Programming (ICALP 81)*, Lecture Notes on Computer Science, 115, pp. 264-277 (Springer Verlag).

Levendel, Y. (1986), "Fault Simulation", in *Fault-Tolerant Computing, Theory and Techniques*, (D. K. Pradhan, Eds.), pp. 184-264 (Englewood Cliffs: Prentice Hall).

Lewis, H. (1990) "A Logic of Concrete Time Intervals", in *Proc. 5th. Symp. on Logic in Computer Science (LICS 90)*, pp. 380-389 (IEEE).

Littlewood, B. (1979), "A Software Reliability Model for Modular Program Structure", *IEEE Transactions on Reliability*, R-28, pp. 241-246.

Littlewood, B. (1987), *Software Reliability Achievement and Assessment*, (Blackwell Scientific Publications).

Littlewood, B. (1988), "Forecasting Software Reliability", in *Software Reliability Modeling and Identification*, (S. Bittanti, Eds.), pp. 140-209 (Springer-Verlag).

Littlewood, B. and Miller, D.R. (1987a) "A Conceptual Model of Multi-Version Software", in *Proc. 17th. Int. Symp. on Fault-Tolerant Computing (FTCS-17)*, Pittsburgh, PA, USA, pp. 150-155 (IEEE).

Littlewood, B. and Miller, D.R. (1987b) "A Conceptual Model of the Effect of Diverse Methodologies on Coincident Failures in Multi-Version Software", in *Proc. 3rd. Int Fault-Tolerant Computer Systems Conf.*, Bremerhaven, Germany (also Centre for Software Reliability Technical Report, May 1987).

Liu, C.L. and Layland, J.W. (1973), "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", *Journal of the ACM*, 20 (1), pp. 46-61.

Loques, O.G. and Kramer, J. (1986), "Flexible Fault Tolerance for Distributed Computer Systems", *Proceedings of the IEE*, E-133 (6), pp. 319-332.

Mahmood, A., Andrews, D.M. and McCluskey, E.J. (1984) "Executable Assertions and Flight Software", in *Proc. 6th. Digital Avionics Systems Conf.*, Baltimore, Maryland, USA, pp. 346-351 (AIAA/IEEE).

Mancini, L.V. and Shrivastava, S.K. (1989) "Replication within Atomic Actions and Conversations: a Case Study in Fault Tolerance Duality", in *Proc. 19th. Int. Symposium on Fault-Tolerant Computing Systems (FTCS-19)*, Chicago, MI, USA, pp. 454-461 (IEEE).

Manna, Z. and Pnueli, A. (1989), "The Anchored Version of the Temporal Framework", in *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, (J. W. De Bakker, W. P. De Roover and G. Rozenberg, Eds.), Lecture Notes on Computer Science, 354, pp. 201-284 (Springer Verlag).

MAP *Network Management Requirements Specification*, MAP/TOP 3.0, Chapter C11 (1987).

Maraninchi, F. (1990) *ARGOS, a Graphical Language for the Design, Description and Validation of Reactive Systems*, Doctoral Dissertation, Université J. Fourier, Grenoble, France (in French).

McCabe, T.J. (1976), "A Complexity Measure", *IEEE Transactions on Software Engineering*, SE-2 (4), pp. 308-320.

McCluskey, E.J. (1986), "Design for Testability", in *Fault-Tolerant Computing, Theory and Techniques*, (D. K. Pradhan, Eds.), pp. 95-183 (Prentice Hall).

Melliar-Smith, P.M. (1987) "Extending Interval Logic to Real Time Systems", in *Proc. Workshop on Temporal Logic in Specification*, Manchester, UK, Lecture Notes in Computer Science, 398, pp. 224-242 (Springer-Verlag).

Melliar-Smith, P.M. and Randell, B. (1977), "Software Reliability: The Role of Programmed Exception Handling", *ACM SIGPLAN Notices*, 12 (3), pp. 95-100 (also in *Reliable Computer Systems*, S.K. Shrivastava (Ed.), Springer-Verlag, 1985, pp.143-153).

Melliar-Smith, P.M. and Schwartz, R.L. (1982), "Formal Specification and Mechanical Verification of SIFT: A Fault-Tolerance Flight Control System", *IEEE Transactions on Computers*, C-31 (7), pp. 616-630.

Mellor, P. (1987), "Software Reliability Modelling: the State of the Art", *Information and Software Technology*, 29 (2), pp. 81-98 (Butterworth Scientific Press).

Meyer, J.F. (1978) "On Evaluating the Performability of Degradable Computing Systems", in *Proc. 8th. Int. Symp. on Fault Tolerant Computing (FTCS-8)*, Toulouse, France, pp. 44-49 (IEEE).

Miller, D.R. (1986a), "Exponential Order Statistic Models of Software Reliability Growth", *IEEE Transactions on Software Engineering*, SE-12 (1), pp. 12-24.

Miller, D.R. (1986b), "Making Statistical Inferences About Software Reliability", *Software Reliability and Metrics Newsletter*, 4, pp. 3 (Center for Software Reliability - Alvey Club).

Milner, R. (1980), *A Calculus for Communicating Processes*, Lecture Notes in Computer Science, 92 (Springer Verlag).

Milner, R. (1983), "Calculii for Synchrony and Asynchrony", *Theoretical Computer Science*, 25 (3), pp. 267-310.

Minet, P. and Sedillot, S. (1987), "Integration of Real-Time and Consistency Constraints in Distributed Databases: The SIGMA Approach", *Computer Standards & Interfaces*, 6 (1), pp. 97-105.

Mishra, S., Peterson, L.L. and Schlichting, R.D. (1989) "Implementing Fault-Tolerant Replicated Objects using Psync", in *Proc. 8th. Symp. on Reliable Distributed Systems (SRDS-8)*, Seattle, Washington, USA, pp. 42-52 (IEEE).

Misra, J. and Chandy, M. (1981), "Proofs of Networks Processes", *IEEE Transactions on Software Engineering*, 7 (4), pp. 417-426.

Molloy, M. (1982), "Performance Analysis using Stochastic Petri Nets", *IEEE Transactions on Computers*, 39 (9), pp. 913-917.

Moreira de Souza, J. and Peixoto Paz, E. (1975), "Fault-Tolerant Clocking System", *Electronic Letters*, 11 (19), pp. 433-434.

Moreira de Souza, J., Peixoto Paz, E. and Landrault, C. (1976) "A Research Oriented Microcomputer with Built-In Auto-Diagnostics", in *Proc. 6th. Fault-Tolerant Computing Symposium (FTCS-6)*, Pittsburgh, PA, USA, pp. 3-8 (IEEE).

Moss, J.E.B. (1981) *Nested Transactions: an Approach to Reliable Distributed Computing*, Massachussetts Institute of Technology.

MP MMS *Specification of a Multipoint MMS*, Delta-4 Implementation Guide, Section 1, Chapter 3 (1991) (available from Bull SA, 1 Rue de Provence, 38423 Echirolles, France).

Myers, G.J. (1979), *The Art of Software Testing*, (John Wiley & Sons).

Nagel, P.M. and Skrivan, J.A. (1982) *Software Reliability: Repetitive Run Experimentation and Modeling*, NASA, Report N°CR-165836.

Navaratnam, S., Chanson, S. and Neufeld, G. (1988) "Reliable Group Communication in Distributed Systems", in *Proc. 8th. Int. Conf. on Distributed Computing Systems (ICDCS-8)*, San Jose - USA, pp. 428-437 (IEEE).

NCSC (1987a) *A Guide to Understanding Discretionary Access Control in Trusted Systems*, National Computer Security Center N°NCSC-TG-003 (Version-1).

NCSC (1987b) *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, National Computer Security Center N°NCSC-TG-005 (Version 1).

Needham, R.M. and Schroeder, M.D. (1978), "Using Encryption for Authentication in Large Networks of Computers", *Communications of the ACM*, 21 (12), pp. 993-999.

Nicolaidis, M., Noraz, S. and Courtois, B. (1989) "A Generalized Theory of Fail-Safe Systems", in *Proc. 19th. Int. Symp. on Fault Tolerant Computing (FTCS-19)*, Chicago, MI, USA, pp. 398-406 (IEEE).

Nicollin, X., Richier, J.-L., Sifakis, J. and Voiron, J. (1990) "ATP, an Algebra for Timed Processes", in *Proc. IFIP WG2.2/2.3 Working Conference on 'Programming Concepts and Methods'*, Sea of Galilee, Israel, pp. 415-442.

Norman, D.A. (1983), "Design Rules Based on Analyses of Human Error", *Communications of the ACM*, 26 (4), pp. 254-258.

Ntafos, S.C. (1988), "A Comparison of Some Structural Testing Strategies", *IEEE Transactions on Software Engineering*, SE-14 (6), pp. 868-874.

Osterweil, L.J. and Fodsick, L.D. (1976), "DAVE — A Validation Error Detection and Documentation System for Fortran Programs", *Software Practice and Experience*, 6 (4), pp. 473-486.

Owicki, S. and Gries, D. (1976), "An Axiomatic Proof Technique for Parallel Programs", *Acta Informatica*, 6, pp. 319–340.

Park, D. (1980) "On the Semantics of Fair Parallelism", in *Proc. Abstract Software Specification*, Lecture Notes in Computer Science, 86, pp. 504-524 (Springer Verlag).

PDCS (1990) *Timeliness*, Specification and Design for Dependability, Esprit Project N°3092 (PDCS: Predictably Dependable Computing Systems), PDCS First Year Report.

Perry, K.J. and Toueg, S. (1986), "Distributed Agreement in the Presence of Processor and Communication Faults", *IEEE Transactions on Software Engineering*, SE-12 (3), pp. 477-482.

Peterson, L.L., Buchholdz, N.C. and Schlichting, R.D. (1989), "Preserving and Using Context Information in Interprocess Communication", *ACM Transactions on Computer Systems*, 7 (3).

Peterson, W.W. and Weldon, E.J. (1972), *Error-Correcting Codes*, (MIT Press).

Pignal, P.I. (1988), "An Analysis of Hardware and Software Availability Exemplified on the IBM 3725 Communication Controler", *IBM Journal of Research and Development*, 32 (2), pp. 268-278.

Plotkin, G.D. (1981) *A Structural Approach to Operational Semantics*, DAIMI, Aarhus, Technical Report N°FN-19.

Pnueli, A. (1977) "The Temporal Logic of Programs", in *Proc. 18th. Symp. on Foundations of Computer Science (FOCS 77)* (IEEE) (Revised version published in *Theoretical Computer Science*, 13:45–60, 1981).

Pnueli, A. (1986) "Specification and Development of Reactive Systems", in *Proc. 10th. IFIP Int. World Computer Congress (Information Processing 86)*, Dublin, Ireland, pp. 845-858 (North-Holland).

Powell, D. (1991) *Fault Assumptions and Assumption Coverage*, LAAS-CNRS, Report N°90.074 (in PDCS Second Year Report, vol.1) (available from LAAS-CNRS, 7 Ave. Colonel Roche, 31077 Toulouse, France).

Powell, D. (Ed.) (1988) *Delta-4 Overall System Specification (OSS)*, (Delta-4 Project Consortium) (Delta-4 Document S88.040/I2/P).

Powell, D., Bonn, G., Seaton, D., Veríssimo, P. and Waeselynck, F. (1988) "The Delta-4 Approach to Dependability in Open Distributed Computing Systems", in *Proc. 18th. Int. Symp. on Fault-Tolerant Computing Systems (FTCS-18)*, Tokyo, Japan, pp. 246-251 (IEEE).

Pradhan, D.K. (1986), "Fault-Tolerant Multiprocessor and VLSI-Based System Communication Architectures", in *Fault-Tolerant Computing, Theory and Techniques*, pp. 467-576 (Prentice Hall).

Puscher, P. and Koza, C. (1989), "Calculating the Maximum Execution Time of Real-Time Programs", *Real-Time Systems*, 1 (2), pp. 159-176.

Queille, J.P. and Sifakis, J. (1983), "Fairness and Related Properties in Transition Systems: a Temporal Logic to deal with Fairness", *Acta Informatica*, 19 (3), pp. 195-220.

Quemada, J., Aczorra, A. and Frutos, D. (1989) "A Timed Calculus for Lotos", in *Proc. 2nd. Int. Conf. on Formal Description Techniques (FORTE 89)* (North-Holland).

Rabin, M.O. (1989), "Efficient Dispersal of Information for Security, Load Balacing and Fault Tolerance", *Journal of the ACM*, 36 (2), pp. 335-348.

Ramamoorthy, C.V., Prakash, A., Tsai, W.T. and Usuda, Y. (1984), "Software Engineering: Problems and Perspectives", *IEEE Computer*, 17 (10), pp. 191-209.

Randell, B. (1975), "System Structure for Software Fault Tolerance", *IEEE Transactions on Software Engineering*, SE-1 (2), pp. 220-232.

Randell, B. (1987), "Design Fault Tolerance", in *The Evolution of Fault-Tolerant Computing*, (A. Avizienis, H. Kopetz and J.-C. Laprie, Eds.), vol. 1 pp. 251-270 (Springer-Verlag).

Randell, B. and Dobson, J.E. (1986) "Reliability and Security Issues in Distributed Computing Systems", in *Proc. 5th. Symp. on Reliability in Distributed Software and Database Systems*, Los Angeles, CA, USA, pp. 113-118 (IEEE).

Randell, B., Lee, P.A. and Treleaven, P.C. (1978), "Reliability Issues in Computer System Design", *ACM Computing Surveys*, 10 (2), pp. 123-165.

Rapps, S. and Weyuker, E.J. (1985), "Selecting Software Test Data using Data Flow Information", *IEEE Transactions on Software Engineering*, SE-11 (4), pp. 367-375.

Rennels, D.A. (1986) "On Implementing Fault-Tolerance in Binary Hypercubes", in *Proc. 16th. Int. Symp. on Fault Tolerant Computing (FTCS-16)*, Vienna, Austria, pp. 344-349 (IEEE).

Ribot, R. (1989) *Specifications of AMp Protocol for the Token Ring*, Delta-4 Document N°W89.125 (available from Bull SA, 1 Rue de Provence, 38423 Echirolles, France).

Richier, J.-L., Rodriguez, C., Sifakis, J. and Voiron, J. (1987) *XESAR: a Tool for Protocol Validation - User Manual*, Laboratoire de Génie Informatique (Edition 1.2).

Risks *Forum on Risks to the Public in Computer Systems*, ACM Committee on Computers and Public Policy, Computer-Distributed Newsletter.

Rivest, R., Shamir, A. and Adleman, L. (1978), "A Method of obtaining Digital Signatures and Public-key Cryptosystems", *Communications of the ACM*, 21 (2), pp. 120-126.

Robinson, D.C. (1988) *Domains - A Uniform Approach to Distributed Systems Management*, Imperial College of Science & Technology, London, UK, Research Report N°88/9.

Rodrigues, L. and Verissimo, P. (1991) *A Posteriori Agreement for Clock Synchronization on Boradcast Networks*, INESC, Technical Report (available from INESC, Rua Alves Redol, 9, 1000 Lisboa, Portugal).

Rodriguez, C. (1988) *System Specification and Validation in XESAR*, Doctoral Dissertation, Institut National Polytechnique, Grenoble, France (in French).

Rook, P. (1990), *Software Reliability Handbook* , (Elsevier Applied Science).

Roth, J.P., Bourricius, W.G. and Schneider, P.R. (1967), "Programmed Algorithms to Compute Tests to Detect and Distinguish between Failures in Logic Circuits", *IEEE Transactions on Electronic Computers*, EC-16 (October), pp. 567-579.

RTCA 178A (1985) *Software Considerations in Airborne Systems and Equipment Certification*, Radio Technical Commission for Aeronautics, Document N°RTCA/D0-178A.

Rudin, H. (1985), "An Informal Overview of Formal Protocol Specification", *IEEE Communications*, 23 (3), pp. 46-52.

Rushby, J.M. and Randell, B. (1983), "A Distributed Secure System", *IEEE Computer*, 16 (7), pp. 55-67.

Rutledge, L.S. (1987) *A Spatial Encoding Mechanism for Network Security*, PhD Thesis, Institute for Information Science and Technology, Washington .

Saltzer, J., Reed, D. and Clark, D. (1984), "End-to-End Arguments in System Design", *ACM Transactions on Computer Systems*, 2 (4).

Schlichting, R.D. and Schneider, F.B. (1983), "Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems", *ACM Transactions on Computer Systems*, 1 (3), pp. 222-238.

Schneider, F.B. (1984), "Byzantine Generals in Action: Implementing Fail-Stop Processors", *ACM Transactions on Computing Systems*, 2 (2), pp. 145-154.

Schneider, F.B. (1986) "A Paradigm for Reliable Clock Synchronization", in *Proc. Advanced Seminar on Real-Time Local Area Networks*, Bandol, France, pp. 85-104 (INRIA).

Schneider, F.B. (1990), "Implementing Fault Tolerant Services using the State Machine Approach: a Tutorial", *ACM Computing Surveys*, 22 (4), pp. 229-319.

Segall, Z., Vrsalovic, D., Siewiorek, D., Yaskin, D., Kownacki, J., Barton, J., Rancey, D., Robinson, A. and Lin, T. (1988) "FIAT — Fault Injection based Automated Testing Environment", in *Proc. 18th. Int. Symp. on Fault-Tolerant Computing (FTCS-18)*, Tokyo, Japan, pp. 102-107 (IEEE).

Sha, L., Lehoczky, J.P. and Rajkumar, R. (1988) *Meeting the Timing Requirements of Distributed Real-time Systems*, Carnegie Mellon University.

Sha, L., Rajkumar, R. and Lehoczky, J.P. (1987) *Priority Inheritance Protocols: An Approach to Real-Time Synchronization*, Carnegie-Mellon University.

Shamir, A. (1979), "How to Share a Secret", *Communications of the ACM*, 22 (11), pp. 612-613.

Shaw, D.E. (1976), "Managing a Software Emergency", *Datamation*, 22 (11), pp. 48-50.

Sheridan, C.T. (1978), "Space Shuttle Software", *Datamation*, 24 (7), pp. 128-131.

Shrivastava, S.K., Dixon, G.D., Hedayati, F., Parrington, G.D. and Wheater, S.M. (1988) *A Technical Overview of Arjuna: a System for Reliable Distributed Computing*, University of Newcastle upon Tyne, Technical Report Series N°262.

Shrivastava, S.K., Ezhilchelvan, P.D., Speirs, N.A., Tao, S. and Tully, A. (1991) *Principal Features of the VOLTAN Family of Reliable Node Architectures for Distributed Systems*, University of Newcastle up Tyne, Technical Report.

Shurek, G. and Grumberg, O. (1990) "The Modular Framework of Computer-Aided Verification: Motivation, Solutions and Evaluation Criteria", in *Proc. Conf. on Automatic Verification (CAV 90)*, Rutgers, NJ, USA, Lecture Notes in Computer Science, 531 (Springer Verlag).

Siewiorek, D.P. and Johnson, D. (1982), "A Design Methodology for High Reliability Systems: The Intel 432", in *The Theory and Practice of Reliable System Design*, (D. P. Siewiorek and R. S. Swarz, Eds.), pp. 621-636 (Digital Press).

Singh, C., Billington, R. and Lee, S.Y. (1977), "The Method of Stages for Non-Markov Models", *IEEE Transactions on Reliability*, R-26 (2), pp. 135-137.

Sloman, M. (1987) *Distributed Systems Management*, Imperial College, Research Report N°87/6.

Sloman, M. (1989) "Domain Management for Distributed Systems", in *Proc. IFIP TC6/WG6.6 Symp. on Integrated Network Management*, Boston, MA, USA (B. Meandzija and J. Wescott, Eds.) (Elsevier Science Publishers BV (North Holland)).

Smith, R.M., Trivedi, K.S. and Ramesh, A.V. (1988), "Performability Analysis: Measures, an Algorithm, and a Case Study", *IEEE Transactions on Computers*, 37 (4), pp. 406-417.

Speirs, N.A. and Barrett, P.A. (1989) "Using Passive Replicates in Delta-4 to provide Dependable Distributed Computing", in *Proc. 19th. Int. Symp. on Fault-Tolerant Computing Systems (FTCS-19)*, Chigago, MI, U.S.A, pp. 184-190 (IEEE).

Srikanth, T.K. and Toueg, S. (1987), "Optimal Clock Synchronization", *Journal of the Association for Computing Machinery*, 34 (3).

Stadler, Z. and Grumberg, O. (1989) "Network Grammars, Communication Behaviours and Automatic Verification", in *Proc. Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, Lecture Notes in Computer Science, 407, pp. 151-165 (Springer Verlag).

Steiner, J.G., Neumann, C. and Schiller, J.I. (1988) "Kerberos: an authentication service for open network systems", in *Proc. USENIX Winter Conference*, Dallas, TX, USA, pp. 191-202.

Strigini, L. (1988) *Support of Design Fault-Tolerance in Delta-4 Software: Proposals*, Delta-4 Project Consortium, IEI Report N°B4-54.

Strigini, L. (1990) *Software Fault Tolerance*, Esprit Project N°3092 (PDCS: Predictably Dependable Computing Systems), PDCS First Year Report.

Strigini, L. and Avizienis, A. (1985) "Software Fault-Tolerance and Design Diversity: Past Experience and Future Evolution", in *Proc. 4th. IFAC Workshop SAFECOMP'85*, Como, Italy, pp. 167-172.

SVC200 *User's Guide to Ferranti Real-Time System V (SVC200)*, Ferranti International, Document N°U20410 (Release 1, Issue 1a) (available from Ferranti International, Wythenshawe, Manchester M22 5LA, UK).

Taylor, D.J. and Black, J.P. (1985) "Guidelines for Storage Structure Error Correction", in *Proc. 15th. Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, Ann Arbor, MI, USA, pp. 20-22 (IEEE).

Taylor, D.J., Morgan, D.E. and Black, J.P. (1980), "Redundancy in Data Structures: Improving Software Fault Tolerance", *IEEE Transactions on Software Engineering*, SE-6 (6), pp. 383-394.

Thatte, S.M. and Abraham, J.A. (1978) "A Methodology for Functional Level Testing of Microprocessors", in *Proc. 8th. Int. Symp. on Fault Tolerant Computing (FTCS-8)*, Toulouse, France, pp. 90-95 (IEEE).

Theuretzbacher, N. (1986) "Using AI-methods to improve Software Safety", in *Proc. 5th. IFAC Workshop SAFECOMP'86*, Sarlat, France (W. J. Quirk, Eds.), pp. 99-105 (Pergamon Press).

Tohma, Y., Tokunaga, K., Nagase, S. and Murata, Y. (1989), "Structual Approach to the Estimation of the Number of Residual Faults Based on the Hyper-Geometric Distribution", *IEEE Transactions on Software Engineering*, SE-15 (3), pp. 345-355.

Trivedi, K.S. (1984), "Reliability Evaluation for Fault-Tolerant Systems", in *Mathematical Computer Performance and Reliability*, (G. Iazeolla, P. J. Courtois and A. Hordijk, Eds.), pp. 403-414 (North-Holland).

Tully, A. and Shrivastava, S.K. (1990) "Preventing State Divergence in Replicated Distributed Programs", in *Proc. 9th. Symp. on Reliable Distributed Systems (SRDS-9)*, Huntsville, AL, USA, pp. 104-113 (IEEE).

Veríssimo, P. (1988) "Redundant Media Mechanisms for Dependable Communication in Token-Bus LANs", in *Proc. 13th. Local Computer Network Conf.*, Minneapolis, USA, pp. 453-462 (IEEE).

Veríssimo, P. (1990) "Real-Time Data Management with Clockless Reliable Broadcast Protocols", in *Proc. Workshop on the Management of Replicated Data*, Houston, Texas, USA (IEEE).

Veríssimo, P. and Marques, J.A. (1990) "Reliable Broadcast for Fault-Tolerance on Local Computer Networks", in *Proc. 9th. Symp. on Reliable Distributed Systems*, Huntsville, AL, USA, pp. 54-63 (IEEE).

Veríssimo, P. and Rodrigues, L. (1990) "Reliable Multicasting in High-Speed LANs", in *NATO Advanced Research Workshop on Architecture and Performance issues of High-Capacity LANs and MANs*, INRIA, Sophia Antipolis, France.

Veríssimo, P., Rodrigues, L. and Baptista, M. (1989) "AMp: A Highly Parallel Atomic Multicast Protocol", in *Proc. SIGCOM'89 Symp.*, Austin, TX, USA, pp. 83-93 (ACM).

Veríssimo, P., Rodrigues, L. and Marques, J. (1987) "Atomic Multicast Extensions for 802.4 Token-Bus", in *Proc. FOC/LAN 87 Conference*, Anaheim, USA.

Voges, U. (1988), *Software Diversity in Computerized Control Systems*, 2 (Springer-Verlag).

Wakerly, J. (1978), *Error Detecting Codes, Self-Checking Circuits and Applications*, (Elsevier North-Holland).

Walker, D.J. (1988) "Bisimulation and Divergence in CCS", in *Proc. 3rd. Symp. on Logic in Computer Science (LICS 88)*, pp. 186-192 (IEEE).

Walter, C.J., Kieckhafer, R.M. and Finn, A.M. (1985) "MAFT: A Multicomputer Architecture for Fault-Tolerance in Real-Time Control Systems", in *Proc. 6th. Real-Time Systems Symp.*, pp. 133-140 (IEEE).

Wensley, J.H., Lamport, L., Goldberg, J., Green, M.W., Levitt, K.N., Melliar-Smith, P.M., Shostack, R.E. and Weinstock, C.B. (1978), "SIFT: The Design and Analysis of a Fault-Tolerant Computer for Aircraft Control", *Proc. IEEE*, 66 (10), pp. 1240-1255.

West, C. (1982) "Applications and Limitations of Automatic Protocol Validation", in *Proc. IFIP WG6.1 2nd. Int. Conf. on Protocol Specification, Testing and Verification*, pp. 361-371.

Weyuker, E.J. (1982), "On Testing Non-Testable Programs", *Computer Journal*, 25 (4), pp. 465-470.

Williams, T.W. (1983), "Design for Testability — A Survey", *Proceedings of the IEEE*, 71 (1), pp. 98-112.

Winskel, G. (1980) *Events in Computation*, PhD Thesis, University of Edinburgh .

Winskel, G. (1984), "Synchronization Trees", *Theoretical Computer Science*, 34 (1-2), pp. 33-82.

Wolper, P. and Lovinfosse, V. (1989) "Verifying Properties of Large sets of Processes with Network Invariants", in *Proc. Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, Lecture Notes in Computer Science, 407, pp. 68-80 (Springer Verlag).

Xu, J. and Parnas, D.L. (1990), "Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations", *IEEE Transactions on Software Engineering*, SE-16 (3).

Yamada, S. and Osaki, S. (1985), "Software Reliability Growth Modeling: Models and Applications", *IEEE Transactions on Software Engineering*, SE-11 (12), pp. 1431-1437.

Yau, S.S. and Cheung, R.C. (1975) "Design of Self-Checking Software", in *Proc. 1st. Int. Conf. on Reliable Software*, Los Angeles, CA, USA, pp. 450-457.

Zafiropulo, P., West, C., Rudin, H., Cowan, D. and Brand, D. (1980), "Towards Analysing and Synthesizing Protocols", *IEEE Transactions on Communication*, Comm-28 (4), pp. 651–661.

Zhao, W., Ramamirithan, K. and Stankovic, J.A. (1987), "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems", *IEEE Transactions on Software Engineering*, SE-13 (5), pp. 564-577.

Ziegler, B.P. (1976), *Theory of Modeling and Simulation*, (John Wiley).