Proc. 6th ESPRIT Conference, Brussels, December 1989.

Project no 1609

SYSTEM MEASUREMENT AND ARCHITECTURE TECHNIQUES (SMART)

M. K. CROWE
Dept of Computing Science
Paisley College of Technology
High Street
Paisley PA1 2BE
Scotland

J. A. CARRASCO
Departament d'Enginyeria Electronica
Universitat Politecnica de Catalunya
Diagonal 647
08028-Barcelona
Spain

Abstract

The SMART project aims to assist the designers of real-time fault-tolerant (RT/FT) systems by providing a number of specially-developed tools within an open environment. The tools developed within the SMART environment support dependability evaluation, fault tree analysis, and simulation of real-time architectures. The SMART environment is open so that designers of RT/FT systems can add their own tools to it.

1. Introduction

The SMART project set out to develop an environment which formalized the evaluation of performance of fault-tolerant systems and to give technical support for their optimisation. In the early stages of the project it was decided to restrict its scope to real-time fault-tolerant (RT/FT) systems.

It used as a main starting point METFAC, a Markovian modelling tool for dependability evaluation developed at the Universitat Politecnica de Catalunya, Barcelona, Spain. Other partners are MATRA-Espace, who are contractors for the data management systems of the main European space applications (Columbus, Hermes, Ariane); CEA, the French nuclear energy authority; CISI-Ingenierie, Paris, France; CRI, Copenhagen, Denmark; and CCS-SCYT, Madrid, Spain, who brought their expertise and approaches to the design of RT/FT systems; while Paisley's contribution has been in tool building.

In common with most ESPRIT projects, the SMART project contributes to the development of European research and its exploitation in industry. Its specific objectives, in relation to fault-tolerant real-time systems, place it at the edge of what is achievable, and the development of research in this area has ruled out as infeasible (within the project timescale), or not useful, a number of approaches that seemed promising at the outset of the project. This article attempts to indicate some of these aspects in addition to documenting what has been achieved.

The SMART project began in May 1987 and is due to end in late 1989. This article gives a view of the project as of September 1989, and so some of the content of this paper deals with work in progress. All the research aspects of the project have been completed, and the remainder of the project has been carefully planned out to be completed on schedule. The development part of the project has resulted in an environment which offers three tools built upon a frame server specially developed in the project. The tools address the areas of dependability evaluation, fault tree analysis, and simulation of real-time architectures. This environment is being demonstrated as part of the ESPRIT Technical Week.

This paper includes synopses of a number of papers internal to the project, which are individually acknowledged. It is a pleasure to thank our colleagues on the project, especially Pascal Paulet (MATRA) and Hans-Kurt Johansen (CRI), for their contributions to the content of this paper; and the anonymous reviewers for their helpful comments.

2. Outline of the SMART project

2.1. Objectives

The SMART project originally intended [1] to assist the design of fault-tolerant systems using a three-dimensional metric system with product, development, and management as axes. An environment was envisaged as supporting data collection, measurement analysis and modelling, to assist system managers and designers in the development of fault-tolerant systems having to meet constraining dependability requirements. It was hoped that the project would be able to use techniques and tools developed in other ESPRIT projects, and that the project would lead to recommendations for the next generation of fault tolerant systems.

It was noted that in nuclear, avionics, and space applications, industry faces a trade-off between high reliability and productivity, with a corresponding need for evaluation and modelling tools. During the early stages of the SMART project, attention was restricted to these areas, which have an important real-time aspect in addition to fault-tolerance, and a data model based on them was constructed.

2.2. Metrics

Metrics, in the sense of numbers associated with systems or components, have an obvious value in classifying or characterising systems. However, in the sense of something that can be calculated automatically from source code, occurrence of faults, or management history, metrics were not found to be helpful in the RT/FT field, which is characterised by low rates of observed faults, small size of software components, and one-off management techniques. For similar reasons, the use of proportional hazards functions and time series, which had been intended alongside METFAC, were found not to be appropriate [2].

A further problem in relation to the original objectives of the SMART project was that the focus on RT/FT systems also made most models for the development environment and the management process break down. For such systems, the development cost is normally a secondary consideration to safety and running costs, and the system designer generally has little interest is management aspects.

2.3. Amended Objectives

At the same time that these negative results were being assimilated, however, it had become clear that important advances could be made by an appropriate redefinition of the objectives of the SMART project. This redefinition of objectives was based on a theoretical study of RT/FT systems, and an assessment of the needs of industry in the RT/FT field. This led to some aspects being selected for completion within the SMART project, and others being highlighted for further work elsewhere.

It was noted that in order to cover the application areas of interest in the SMART project, including avionics, space missions, and nuclear safety, both non-repairable and repairable systems should be considered. For some systems, design support can be provided for the average user on a safe theoretical basis. For others, the dependability models are so complex that the tools can only be put at the disposal of an expert user. The following section describes the theoretical principles underlying this classification [3].

3. General Structure of Real-Time Fault-Tolerant Systems

3.1. Dependability and Coverage

In considering RT/FT systems, the key concepts are dependability and coverage[4]. Dependability deals with the factors that justify faith in the system's ability to perform according to expectations, and includes such attributes as reliability, safety and availability. Coverage is the probability that the system will remain operational following a given exceptional condition; such as component failure.

In many application areas, the normal way to obtain a high dependability system from hardware components is to arrange them in a parallel processing configuration with redundancy. Typically, the computer system receives input data from the process to be monitored using N-fold redundant sensors and produces a set of corresponding analog or digital output signals. These N signals enter a majority voting device which produces the final value output by the process. In cases of unacceptable disagreements between corresponding values a fault/error handling mechanism is activated. This mechanism is responsible for identifying and isolating faulty components and reconfiguring the system in a gracefully degraded mode of operation.

3.2. Numerical Difficulties

In RT/FT systems the execution of application tasks is organised in control loops with a cycle time of the order of tens of milliseconds. A task that fails to complete in that time is considered to have failed, and any output from it is ignored. The fault/error handling mechanisms mentioned above therefore have to be fast since the operation of the system cannot be stopped for recovery. This implies that recovery actions are typically several orders of magnitude faster than failure and repair actions.

Systems of the type considered in the SMART project need to be ultra-reliable. For example, international airflight regulations require that a failure of the system must occur with a rate smaller than 108-99 per hour for a 10 hour aircraft flight. When failure rates have to be as small as this, it is impossible to determine their values by testing, and mathematical methods are required instead. Because the applications are in safety-critical systems, the objective of dependability analysis is to prove that the dependability calculation is based on conservative models of the system, and known conservative approxi-

mation methods are used to model fault/error behaviour and handling (see, for instance 5). High precision computations on large Markov models are not necessarily required, particularly since the transition rates are not known precisely. On the other hand, a sensitivity analysis on input data such as component failure rates is required. Getting good values for the model parameters is costly, and sensitivity analysis can be used to establish acceptable error margins for the various parameters, in relation to the size of their contribution to the metric of interest.

3.3. Decomposition Techniques

In fact, the difference of orders of magnitude between the durations associated to recovery actions and failure/repair actions, which is typical of RT/FT systems, allows the use of the behavioural decomposition technique[6], which has been proved to give conservative estimates, and fortunately, resolves many of the numerical difficulties mentioned earlier[7]. In this technique, the behaviour of the system is modelled using a fault occurrence and repair model (FORM) and a number of fault/error handling models (FEHM). The FEHMs are used to estimate fault coverages, which are incorporated in the FORM as instantaneous switching probabilities following fault actions.

The development of tools for specification and solution of FEHMs has been relegated in SMART to further work. Nevertheless, the simulation tools provided by the SMART project for interactively analysing the mechanisms for dealing with errors are of considerable value within the design and management teams of a project.

The tool dedicated to FORM specification and solution in the SMART environment is an enhanced version of METFAC. A hierarchical high-level language for describing the behaviour of fault-tolerant repairable systems has been developed as an alternative to production rules, and a FORM generator based on that language has been developed and integrated with the original version of the tool. This language is described in the next section.

4. Dependability Evaluation

4.1. Metfac

Dependability evaluation is supported in the SMART environment by an enhanced version of METFAC[8]. METFAC is a tool for specification and solution of Markovian dependability models using homogeneous continuous-time Markov chains. For such models, METFAC computes a number of dependability metrics including reliability, availability, mean time to first failure, as well as performance and cost-related metrics. The main enhancement to the tool implemented in the context of the SMART project has been a new more user-friendly front end for model specification based on a high-level language [9], covering occurrence, propagation and repair of software and hardware faults in hierarchical systems. This high-level language can be used for a wide variety of fault-tolerant repairable systems, but as certain aspects of the modelling process have been built in to the language, thus restricting the class of models for which the high-level language is appropriate, the original front end, based on production rules, has been retained as an alternative.

4.2. The Form Language

The high-level language takes the SAVE [10] modelling language as its starting point but adds new features which expand significantly both the user-friendliness and modelling power of the language used in SAVE. The main features added are: support for hierarchical model specification through the cluster concept, parameterisation of component and cluster classes, and support for the specification of systems with several operational configurations which can be used successively as resources fail. The introduction of the cluster concept not only allows concise specifications of complex systems, but supports the reduction of the state space through use of aggregation techniques.

Experiments using the language have shown its power and facility in describing fault-tolerant computer systems. For example, the Ariane system [11], a case-study proposed by MATRA, contains 54 components which would have to be considered different in a flat, component-level description, whereas using the hierarchical language it is sufficient to describe only two cluster classes and four component classes.

Rather than give the full syntax of the language, its flavour can perhaps best be illustrated by an extract from this example (Figure 1). The extract shows of the system specification, which follows the definition of some system parameters and constants (such as the failure rate "mcufr" below), and is followed by the specification of cluster classes, component classes, and repair teams.

The language also allows the modelling of error propagation, and of repair strategies assigning priorities to repairing failed components. The language was developed at UPC together with the specification of the FORM generation algorithms. Paisley developed a special-purpose window-based editor and preprocessor for the language; and SCYT developed the FORM generator.

5. Safety Analysis

For the purposes of the SMART project, safety is defined as a measure of the time to catastrophic failure. Safety related specifications are generally expressed using requirements such as Fail Operational (FO) or Fail Safe (FS), or indicating fail-safe only on the second fault (e.g. FO/FS).

The use of timed Petri nets [12] is unpractical owing to the size of the Petri nets involved. Two approaches are available:

(a) failure mode effect analysis (FMEA), which is qualitative in nature, and helps to identify critical components;

(b) fault tree analyser, which obtains estimates for the unreliability of the system associated to each minimal cut in the fault tree.

Tools for performing the fault tree analysis exist. The contribution of the SMART project will be to provide means of combining the fault trees of components to enable the fault tree analysis of the system to be done without first generating the expanded fault tree of the system.

FIGURE 1 ARIANE

```
System
    Made of
        1 Mt of TC(mcufr, mcuc)
        1 S1t of TC(s1cufr,s1cuc)
        2 S2t of TC(s2cufr,1)
        5 S3t of TC(s3cufr,1)
        1 Bus1 of BusC
        1 Bus2 of BusC
        1 Recovery of RecoveryC
    Resource Attributes
        Can_by_B1: Bus1[1] and Mt.Can_by_B1[1] and S1t.Can_by_B1[1]
            and S2t.Can_by_B1[2] and S3t.Can_by_B1[5]
        Can_by_B2: Bus2[1] and Mt.Can_by_B2[1] and S1t.Can_by_B2[1]
            and S2t.Can_by_B2[2] and S3t.Can_by_B2[5]
        Can_by_B1_and_B2: Bus1[1] and Bus2[1] and
            Mt.Can_by_B1_and_B2[1] and
            S1t.Can_by_B1_or_B2[1] and
            S2t.Can_by_B1_or_B2[2] and
            S3t.Can_by_B1_or_B2[5]
    Operational Requirements
        Recovery[1] and (Can_by_B1 or Can_by_B2 or Can_by_B1_and_B2)
    Configuration
        Requirements: Can_by_B1
        Clusters: TC
            Using_B1: all
        Components: BusC, Recovery
            Operational: all
    Configuration
        Requirements: Can_by_B2
            Clusters: TC
        Using_B2: all
            Components: BusC, Recovery
        Operational: all
    Configuration
        Requirements: Can_by_B1_and_B2
        Clusters: Mt
            Using_B1_and_B2: 1
        Clusters: S1t, S2t, S3t
            Using_B1_or_B2: all
        Components: BusC, Recovery
            Operational: all
```

6. Real-Time Architecture Simulation

The tool described in this section is being developed as part of the SMART project to assist the designers and validators of RT/FTS. Interaction with the display and the user is by means of the X Window System [13].

6.1. Architecture Description

The real-time architecture simulator (RTAS) provides a method of system description

which takes into account the two complementary aspects of a system: architectural and behavioural. The architectural aspect describes the construction of the system from components (modular objects), which may be hardware or software. The behavioural aspect describes the evolution of the system in time from one state to the next. In principle the modelling can take place at any level of design detail, but in practice it is most useful on models of the global design of the system.

The architectural description includes both logical and graphical information. The logical information describes the relationships (connections) between components of the architecture, while the graphical information describes how the model can be displayed for the user (creation of a block diagram). The RTAS can be commanded to display a synoptic of the system from the views of the different components during the simulation. A graphical editor is provided to enable the user to modify the appearance of a displayed object.

The behavioural description can deal with error states and error handling in addition to the nominal behaviour of the system: a timed transition high level Petri net formalism is used. Graphical and textual editors are provided for such Petri nets.

6.2. Using The Simulator

Interaction with the RTAS allows an initial state to be set up, and the system behaviour set in motion for a given number of steps or time interval. System evolution is shown on the display by using highlighting, updating variables, etc. No performance is measured, but the real-time functionality can be checked using tracing facilities, animated block diagrams and chronograms.

In this way the system designer can show the nominal behaviour of the system, and can intervene to inject events into the system, by setting variables tested by transition predicates in the Petri nets. Consequently, the FEHM behaviour can be explored, enhanced, and to some extent validated: both hardware and software faults can be analysed, depending on the level of detail provided by the model.

Because fault scenarios can be complex, tracing and some other monitoring facilities are available. This helps when an interesting behaviour mode needs further analysis.

6.3. Implementation

The architecture of the SMART environment is such that the RTAS is implemented as a set of processes co-operating through the medium of the X server and the SMART frame server, which is described in the next section.

A components library for grouping the user models is supported, with a pictorial high level language and synoptic binding in addition to description of operation using a programming language (C). This allows modelling by simple Petri nets with complicated operations, even referring to external C code, as an important alternative to detailed Petri nets.

Systems such as RTAS are frequently implemented using Smalltalk [14]. However, the disadvantage of Smalltalk is that it does not integrate well with Unix [15], and it was considered important to develop an open system in the SMART project, that could be integrated at a later stage with other developments such as IPSEs [16], the PCTE [17] etc.

7. The Frame Server

This section describes the SMART frame server [18], which has been implemented as part of the SMART project.

The frame server provides the infrastructure of the SMART environment. The concept of frame used in the SMART project is rather different from that of Minsky [19], but similarly provides an extremely general method of describing objects and classes. For example, a model description in the FORM language, as described above, is a single frame, as illustrated below. The FORM language defines what attributes should be expected in such a frame.

The server manages a set of independent contexts on behalf of client applications. In each context, a set of frames forms a semantic network which can be navigated, consulted or updated by one or more clients. The use of the frame server minimises access to the disk, and allows a client to be notified if an item of information of interest to that client is updated.

7.1. Frames

Frames have types, but the semantics of the frame type is defined by another frame (usually a Sublanguage-specification frame). The semantics of a frame type is context-dependent, since it depends on which sublanguages are defined in the context's semantic network, and it is possible for a given frame to appear to have different contents in different contexts. There are a number of predefined frame types, including Sublanguage-specification, Context, and Window-data: a window-data frame can be used to describe the contents of a window on the display.

Frames contain attribute/value lists: by default an attribute value consists of text. Repeating attributes are supported, and attributes may have sub-attributes.

For example, representing sub-attributes by indentation, the FORM fragment given earlier corresponds to the frame portion in Figure 2.

7.2. Semantics

Attributes may have associated hook functions, which leads to a kind of access-oriented programming. An important hook function is make-link, which means that the attribute value is the name of a frame to be installed at that point in the semantic network. Methods associated with a frame type allow the user to define new hook functions: and frame types can specialise other frame types, inheriting their methods and top-level attributes, so that object-oriented programming is supported too.

A transaction mechanism supports graceful error recovery and preserves consistency during complex updates to the semantic network. Syntactic and lexical definitions allow the frame server to select portions of attribute values.

Frames on disk can be used to provide a starting state for the semantic network. Window-based browsing and editing tools are provided. A memory-resident frame for each context allows access to internal information about that context, such as the identities of any frames that have been modified but not written to disk.

It follows from the above description of the frame server that the data model in SMART is open. New attributes can be added to existing frame types without any change being needed to existing software. New views of existing frames can be provided simply by using a modified sublanguage-specification. This makes development of the model a simple matter.

Figure 2 A Frame Example

Attribute	Value
System	
Made_of	
Pieces	1 Mt of TC(mcufr, mcuc)
Pieces	1 S1t of TC(s1cufr,s1cuc)
Pieces	2 S2t of TC(s2cufr,1)
Pieces	5 S3t of TC(s3cufr,1)
Pieces	1 Bus1 of BusC
Pieces	1 Bus2 of BusC
Pieces	1 Recovery of RecoveryC
Resource_Attribute	Can_by_B1: Bus1[1] and Mt.Can_by_B1[1] and S1t.Can_by_B1[1] and S2t.Can_by_B1[2] and S3t.Can_by_B1[5]
Resource_Attribute	Can_by_B2: Bus2[1] and Mt.Can_by_B2[1] and S1t.Can_by_B2[1] and S2t.Can_by_B2[2] and S3t.Can_by_B2[5]
Resource_Attribute	Can_by_B1_and_B2: Bus1[1] and Bus2[1] and Mt.Can_by_B1_and_B2[1] and S1t.Can_by_B1_or_B2[1] and S2t.Can_by_B1_or_B2[2] and S3t.Can_by_B1_or_B2[5]
Operational_Requirements	Recovery[1] and (Can_by_B1 or Can_by_B2 or Can_by_B1_and_B2)
Configuration	
Requirements	Can_by_B1
Clusters	TC
Mode	Using_B1: all
Components	BusC, Recovery
Mode	Operational: all
Configuration	
Requirements	Can_by_B2
Clusters	TC
Mode	Using_B2: all
Components	BusC, Recovery
Mode	Operational: all
Configuration	
Requirements	Can_by_B1_and_B2
Clusters	Mt
Mode	Using_B1_and_B2: 1
Clusters	S1t, S2t, S3t
Mode	Using_B1_or_B2: all
Components	BusC, Recovery
Mode	Operational: all

7.3. Implementation

Frames on disk are implemented as text files with the names of attributes stored along with their values, and the hierarchical structure being controlled by special characters. This is an efficient use of disk space since a single frame can contain a substantial amount of information. When a frame is read into a context, the frame type controls the building of efficient data structures in the server's memory which facilitate access to the frame contents. Linked frames are installed in these structures so that their contents appear as sub-attributes of the links.

Clients of the frame server issue commands and queries to the server: just as for data bases, the queries include the names of attributes requested. The corresponding retrieval is retained by the server to be the subject of further requests by the client (e.g. update, insert, delete).

The server-client communication is built on the reliable transport layer of the Unix inter-process communication system, and is similar to that of the X Window System.

8. Conclusions and suggestions for further work

The SMART project has been successful in devising an environment of use to the designers of real-time fault-tolerant systems. It has brought a number of theoretical results on dependability into the mainstream of industrial exploitation.

8.1. Further Research

Further work in the dependability area is required into

8.1.1. FEHM. The study of general exit time distributions for FEHM, possibly leading to a kind of importance sampling technique in the simulation algorithm;

8.1.2. Metric evaluation. Following up suggestions in the literature for evaluation of dependability and related metrics [20,21,22];

8.1.3. Sensitivity analysis. Multi-parameter sensitivy analysis using eigenvector analysis of a locally linearised matrix of partial derivatives;

8.1.4. Multiphased systems. Development of numerical methods for the evaluation of metrics for multiphased systems;

8.1.5. Approximation Techniques. Development of approximation techniques for FORM's with provable error bounds.

8.2. Tool Enhancements

8.2.1. METFAC.

The SMART project is making the METFAC dependability evaluation tool more accessible to industrial developers. A number of proposed enhancements to METFAC would develop from the research areas mentioned above: application to multiphased systems, and approximation methods to reduce the state space.

8.2.2. RTAS.

The SMART real-time architecture simulator supports an integrated approach to system simulation and validation of fault/error handling methods. It would be convenient to add the standard verification algorithms for Petri nets (cyclicity, liveness, boundedness) and facilities for verifying invariants. Other ways in which the RTAS could be developed would be to generalise the user interaction and display methods so that quite general systems could be animated.

8.2.3. Frame Server.

The frame server concept appears to have a usefulness in a number of other areas, and its potential use is currently being explored in a number of research projects. Further interfaces with databases and windowing systems could be developed so that it could be used in other contexts, and a number of optimisations would be desirable, such as more efficient use of virtual memory.

8.3. Evaluation

Some validation studies are already in progress as part of the SMART project, and further evaluation studies, possibly coupled with some of the above suggestions for further work, may find a place in the ESPRIT programme. It is expected, however. that an industrially-significant toolset to help with the design of RT/FT systems will emerge from the SMART project, and therein will lie its success.

References

1. Kuntzmann, A. (1986) 'SMART: System Measurement and Architecture Techniques', ESPRIT project proposal.

2. Musa, J. D. (1987) 'Software Reliability Measures: Guiding Software Development for Cost-Effective System Quality', ESEC '87 1st Europoean Software Engineering Conference (Tutorial 2).

3. Johansen, H. K. (1988) 'Dependability Evaluation of Real-Time Fault-Tolerant Systems', SMART working paper, CRI Copenhagen.

4. Laprie, J. C. (1985) 'Dependable Computing and Fault-Tolerance: Concepts and Terminology', Proc. 15th Int Symp on Fault-Tolerant Computing, pp. 2-11.

5. Wensley, J. H. (1978) 'SIFT: Design and Analysis of a Fault-tolerant Computer for Aircraft Control', Proceedings of the IEEE, vol. 66, no. 10, pp. 1240-1255.

6. Geist, R. M. and Trivedi, K. S. (1983) 'Decomposition in reliability analysis of fault-tolerant systems', IEEE Transactions on Reliability, vol. R-32, pp. 463-468.

7. McGough, J., Smotherman, M., and Trivedi, K. S. (1985) 'The conservativeness of reliability estimates based on instantaneous coverage', IEEE Transactions on Computers, vol. C-34, pp. 602-609.

8. Carrasco, J. A. and Figueras, J. (1986) 'METFAC: Design and Implementation of a Software Tool for Modelling and Evaluation of Complex Fault-Tolerant Computing Systems', Proc 16th Int Symp on Fault-Tolerant Computing, pp. 424-429.

9. Carrasco, J. A. (1989) 'A high-level modelling language for fault-tolerant computer systems', SMART working paper, UPC Barcelona.

10. Goyal, A., Carter, W. C., de Souza e Silva, E., Lavenberg, S. S., and Trivedi, K. S. (1986) 'The

System Availability Estimator', Proc 16th Int Symp on Fault-Tolerant Computing, pp. 84-89.

11. Sotta, J. P. (1988) 'Modelling the Ariane System', SMART working paper.

12. Leveson, N. G. and Stolzy, J. L. (1985) 'Safety Analysis using Petri Nets', Proc 15th Int Symp on Fault-Tolerant Computing.

13. Fulton, J. (1988) X Window System, Version 11, X Consortium, MIT Laboratory for Computer Science.

14. Goldberg, A. (1983) 'The Influence of an Object-Oriented Language on the Programming Environment', ACM Computer Science Conference, pp. 35-44, Orlando, Florida.

15. Ritchie, D. M. and Thompson, K. (1978) 'The Unix Time Sharing System', The Bell System Technical Journal, vol. 56, no.6, pp. 1905-1929. Unix is a trademark of AT&T Bell Laboratories.

16. US Dept of Defense (1980) 'Requirements for Ada Programming Support Environments', STONEMAN.

17. ESPRIT (1985) PCTE: A Basis for a Portable Common Tool Environment. (Functional Specification)

18. Crowe, M. K. and Oram, J. W. (1989) 'An Applicationlevel Frame Server', SMART Technical report, Paisley College of Technology.

19. Minsky, M. (1980) 'A framework for representing knowledge', in Mind design: Philosophy, psychology, and artificial intelligence, ed. J Haugeland, MIT Press, Cambridge, Mass.

20. Heidelberg, P. and Goyal, A. (1987) 'Sensitivity Analysis of Continuous Time Markov Chains using Uniformization', Proc of the 2nd Int Workshop on Applied Mathematics and Performance/Reliability Models of Computer/Communication Systems, Rome.

21. Singh, C., Billington, R., and Lee, S. Y. (1977) 'The Method of Stages for Non-Markov Models', IEEE Transactions on Reliability.

22. Conway, A. E. and Goyal, A. (1987) 'Monte Carlo Simulation of Computer System Availability/Reliability Models', Proc 17th Int Symp on Fault-Tolerant Computing, pp. 230-235.