Predictably Dependable Computing Systems: Second Year Report

B. Randell and J-C Laprie

Abstract

Predictably Dependable Computing Systems (PDCS) is an ESPRIT Basic Research Action involving eight institutions. These are: The City University (London), IEl del CNR (Pisa), Universität Karlsruhe, LAAS-CNRS (Toulouse), The University of Newcastle upon Tyne, Université Paris-Sud, Technische Universität Wien, and The University of York. The Action involves approximately 40 staff at the participating institutions. It started formally in May 1989, and is due to last in the first instance until October 1991. The objective of the PDCS Project is to contribute to making the process of designing and constructing dependable computing systems much more predictable and cost-effective than it is at present. To this end, the Project has been conducting research into (i) unifying concepts underlying dependability, (ii) means for the establishment and validation of dependability requirements, (iii) stochastic techniques for assessing and predicting dependability. The ultimate long term objective is to produce a design support environment which is well-populated with tools and ready-made system components and which fully supports the notion of predictably dependable design of large real-time fault-tolerant distributed systems.
Bibliographical details

RANDELL, Brian
Predictably Dependable Computing Systems : Second Year Report
[By] B. Randell and J-C Laprie
Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Laboratory, 1991.
(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 343)

Added entries
UNIVERSITY OF NEWCASTLE UPON TYNE.
Computing Laboratory. Technical Report Series. 343
LAPRIE, Jean-Claude

Abstract

Predictably Dependable Computing Systems (PDCS) is an ESPRIT Basic Research Action involving eight institutions. These are: The City University (London), IEI del CNR (Pisa), Universität Karlsruhe, LAAS-CNRS (Toulouse), The University of Newcastle upon Tyne, Université Paris-Sud, Technische Universität Wien, and The University of York. The Action involves approximately 40 staff at the participating institutions. It started formally in May 1989, and is due to last in the first instance until October 1991. The objective of the PDCS Project is to contribute to making the process of designing and constructing dependable computing systems much more predictable and cost-effective than it is at present. To this end, the Project has been conducting research into (i) unifying concepts underlying dependability, (ii) means for the establishment and validation of dependability requirements, (iii) stochastic techniques for assessing and predicting dependability. The ultimate long term objective is to produce a design support environment which is well-populated with tools and ready-made system components and which fully supports the notion of predictably dependable design of large real-time fault-tolerant distributed systems.

About the author

Professor Randell has been a professor of computing science at the Computing Laboratory of the University of Newcastle upon Tyne since 1969.

Dr. Laprie is a "Directeur de Recherche CNRS" at LAAS where since 1975 he has directed the Fault Tolerance and Dependable Computing research group.
Suggested keywords

DEPENDABILITY
FAULT TOLERANCE
HARDWARE
MODELLING AND EVALUATION
PREDICTION
SOFTWARE

Suggested classmarks (primary classmark underlined)
Dewey (18th): 001.64404  621.381958
U.D.C. 519.687  681.322.022
Predictably Dependable Computing Systems: Second Year Report

Brian Randell

Computing Laboratory
University of Newcastle upon Tyne, U.K.

Jean-Claude Laprie

LAAS-CNRS
Toulouse, France

Introduction

Predictably Dependable Computing Systems (PDCS) is ESPRIT Basic Research Action 3092. The institutions and principal investigators involved in PDCS are:

Centre for Software Reliability, The City University, London, UK (Bev Littlewood)

IEI del CNR, Pisa, Italy (Lorenzo Strigini)

Institut fur Algorithim und Kognitiv Systeme, Universität Karlsruhe, Karlsruhe, Federal Republic of Germany (Tom Beth)

LAAS-CNRS, Toulouse, France (Jean-Claude Laprie)

Computing Laboratory, The University of Newcastle upon Tyne, Newcastle upon Tyne, UK (Brian Randell)

LRI, Université Paris-Sud, Paris, France (Marie-Claude Gaudel)

Institut fur Technische Informatik, Technische Universität Wien, Vienna, Austria (Herman Kopetz)

Department of Computer Science, The University of York, York, UK (John McDermid)

The coordinating contractor is the University of Newcastle upon Tyne. Brian Randell is PDCS Project Director, who co-chairs the Project's Executive Board with
Jean-Claude Laprie. The project's Technical Coordinator is John Dobson. The Action involves approximately 40 staff at the participating institutions. It started formally in May 1989, and is due to last in the first instance until October 1991.

The objective of the PDCS Project is to contribute to making the process of designing and constructing dependable computing systems much more predictable and cost-effective than it is at present. To this end, the Project has been conducting research into (i) unifying concepts underlying dependability, (ii) means for the establishment and validation of dependability requirements, (iii) stochastic techniques for assessing and predicting dependability. The ultimate long term objective is to produce a design support environment which is well-populated with tools and ready-made system components and which fully supports the notion of predictably dependable design of large real-time fault-tolerant distributed systems.

This paper summarizes the work that has been done during the project, particularly during its second year, and is in fact based on the summaries provided in the various chapters of our Second Year Report to the CEC, the contents listing of which is given in Appendix 2. Many of the detailed sections of this, and the First Year Report, have appeared as PDCS Technical Reports, and have been (or are due shortly to be) published in various refereed technical conference proceedings and journals. (Appendix 2 gives details of these publications, and of other publications that report on work that has been performed in whole or in part within PDCS.)

Anyone wishing to be put on the PDCS mailing list and receive the Technical Reports as they are published, should write to Nick Cook, Computing Laboratory, University of Newcastle upon Tyne, Newcastle NE1 7RU, U.K. (email: Nick.Cook@newcastle.ac.uk).

1. Dependability Concepts

The Project's work on the underlying concepts of dependability is viewed as being of fundamental importance to PDCS. Members of PDCS have been instrumental in the development of the concepts and terminology of dependability through involvement in IFIP WG10.4 and other fora. The First Year Report included a version of the definitions developed in IFIP WG 10.4 and extended by other members of the project. A number of incremental changes and improvements have been made to the document, but the bulk of the material remains unchanged. (It will be published shortly, together with French, German, Italian, and Japanese versions, as volume 5 of the Springer-Verlag series "Dependable Computing and Fault-Tolerant Systems".) However work has gone on in three specific areas which are summarized below.

The first key area we have been examining is how to extend the basic concepts and vocabulary of dependability so as to express concerns of 'unfitness-for-purpose' (which is not the same as 'failure'). The problem is that whereas the notions of error, fault, failure and so on are, at least in principle, easily characterized in terms of such things as component state and service delivery (e.g. an error might be defined as a component being in a state which a specification says it should not be in), these
underlying concepts of state and service delivery do not relate to unfitness-for-purpose in anything like the same way, if indeed at all. For example, if an IT system is intended to reduce staffing in a department by 10% and actually increases it by 50%, it is clearly unfit for its purpose. However, the consequential staff increase is hardly to be considered as part of the service delivery of the IT system, or indeed of any system; nor is it sensible to ask which component of which system was in an error state, though clearly the system as a whole has failed (or ‘failed’) in the sense that it has failed to satisfy its owner’s expectations - a failure which of course can be traced back to a failure in the system which generated the IT system.

The second key area concerns safety and security, which are important subclasses of dependability. The main set of dependability definitions [Laprie, 1990] makes distinctions between these two terms, but the problem of analyzing ‘situations’ and deciding unambiguously which are safety-related and which are, or are also, security-related is not fully solved. Our work has investigated a number of underlying distinctions (a) in the nature of the harm that can be caused by a misbehaving system, and (b) in the nature of the causal chain leading up to and from the actual harmful event. These underlying distinctions would seem to be more important than the categorization of a particular harmful event in the safety/security dimension, since (a) understanding the nature of the harm governs decisions about such things as insurance, compensation, liability, blame, retribution and so on; and (b) understanding the causal chain governs decisions about whether it is desirable and feasible to deploy preventative measures and, if so, which ones; and it is these decisions that are important.

The third key area is that of measurement. It is not meaningful to talk about predictably dependable computing systems unless we have some basis on which to measure, and hence validate, our predictions. Dependability poses some significant problems from the perspective of measurement theory, as many dependability properties are relations between systems and their environment, not properties of the system in vacuo. Further there are fundamental difficulties in determining the nature of the measurement scale which is appropriate for many of the properties. These issues are dealt with in a series of papers, which have recently been published in book form.

2. Effects Of Operational Environments On Dependability

There is general agreement that the ultimate dependability of software during operation is determined not only by the intrinsic properties of the software itself but also by factors of its operational environment: the same software executed in two distinct environments will exhibit different dependability in each. As a particular facet of this general problem, the project has worked on possible means of modelling the effect of operational environment on the particular dependability attribute of reliability.

The most popular method of estimating the reliability of software has been to model reliability as a stochastic point-process of failure events along an execution time axis. Theoretical models have enabled estimates of current and future reliability to be derived using just the so-far-observed part of this process for the single software item
under study. Our recent work has concentrated on the idea of extending the sources of data input to such probabilistic models.

Our initial focus has been on the choice of probabilistic model - it is suggested that certain general statistical regression models which have previously found application in diverse problem areas such as hardware reliability, insurance, and medicine may be capable of discovering and confirming relationships between current software reliability and explanatory data which encompasses more than just the past reliability behaviour of the single piece of software in question. These models have the advantages of flexibility and of amenability to techniques of model fitting and validation which are already highly developed following application over some years in other contexts.

The Project has surveyed previous work on explaining software reliability behaviour in terms of various kinds of other data. Several diverse sources of data have been considered, leading to the suggestion that of these the greatest promise of success is to be attained by choosing different operational environments of a single item of software as the statistical 'individuals' in the statistical model, with measures of one or more attributes of each member of a sample of such individuals as explanatory data for investigating reliability variation over the sample.

If such explanatory factors of reliability can be identified and rigorously validated the benefits are obvious both for research into techniques of controlling reliability as well as for practical reliability prediction. However, the data requirements for empirical validation and exploitation of such relationships are significant. Both observed reliability data and also data capturing something of the variation between different operational environments of a single software product would be required.

3. Specification And Design For Timeliness

At present, two fundamentally different paradigms for the design of real-time systems are being pursued by various different groups. An event-based design focuses on the occurrences of events in the controlled object to initiate the appropriate system activities. A state-based or time-triggered design is driven by the periodic observation of the state of the controlled object and starts actions as a consequence of the analysis of these states.

The Project has analyzed the potential and the limitations of these two approaches. In order to establish a common basis for discussion, a reference model of a distributed real-time system architecture has been produced. This model is based on realistic assumptions about the availability of a global time-base. Real-time entities, denoting the relevant state-variables in the environment, and real-time objects for representing the real-time entities within the computer system are introduced. Information stored in such an object is termed (temporally) "accurate", if the real-time interval since its point of observation is shorter than a given application specific parameter, and "consistent" if it matches that in replicated copies of the object held at other nodes.
Using this model the effects of event-triggered and time-triggered architectures and communication protocols on the accuracy and consistency of the information stored in real-time objects have been investigated. The results of these investigations are expressed in a set of lemmas concerning the timeliness properties of these two architectural approaches. These results are counter intuitive. Whereas one would assume that an event-based architecture has superior temporal properties than a time-triggered architecture, the opposite can be true, particularly in a peak load scenario.

Although time-triggered architectures are more predictable than event-triggered architectures, they lack flexibility. The maximum execution time of all real-time tasks must be analyzed at compile time and all execution schedules have to be prearranged before the system is put into operation. A second aspect of the Project’s work on real-time systems is aimed at overcoming some of these limitations of time-triggered architectures. The need for flexibility, in response to levels of non-determinacy in the system’s environment often leads to the use of dynamic, or even adaptive, run time systems. These systems gain their flexibility at the cost of execution overhead and lack of predictability. We have tried to gain some of this flexibility while still maintaining a high degree of predictability. This is achieved by introducing, in addition to the periodic tasks handled by the time-triggered architecture, a new task set, the sporadic tasks. In order to guarantee the timeliness of sporadic tasks, a minimum time-interval between successive task activations must be assumed. It is then possible to consider the sporadic task arrivals in the design of the time-triggered schedules.

Another form of flexibility can be achieved by statically planning a set of schedules, one for each operational mode of an application, and performing a mode change dynamically during the operation of the system. This topic of mode changes is an important and challenging problem, which requires considerable further investigations.

As a consequence of our investigations we have gained a much clearer picture about the advantages and limitations of event-triggered and time-triggered architectures. In complex distributed real-time applications there is the need for both approaches. If the timeliness of a set of tasks has to be guaranteed by design, time-triggered architectures are the only alternative in distributed systems. Active redundancy can also be implemented much more easily in this type of system. However, if the application requires a high degree of flexibility, e.g., the execution sequence of a real-time transaction depends dynamically on the data values accessed, then an event-triggered architecture is the only choice. However, according to the present state of the art, no analytical timeliness guarantees can be provided in such an approach. Our future research will also focus on providing appropriate interfaces between time-triggered and event-triggered systems.

4. Dependability Requirements

The Project has studied the problem of so-called 'non-functional requirements' (hereinafter referred to as "NFRs") such as security, user-friendliness, growth potential, maintainability, etc., with a view to providing some taxonomic principles for
the classification of NFRs so that they could more easily be expressed in procurement specifications and so that procured systems could be validated against them. The starting point was a lengthy, but still non-exhaustive, list of typical NFRs.

At the outset of the study it was recognized that the distinction between 'functional' and 'non-functional' requirements is a very difficult one to make in practice, and not very useful because:

(i) A requirement goes through a number of transformations before being embodied in the final system, and this is just as much true of so-called 'functional' ones as it is of so-called 'non-functional' ones. It is not easy to see a major difference in requirements based on the structure of this transformation process.

(ii) There is a misleading implication that 'functional requirements' are seen as primarily behavioural, whereas 'non-functional' ones are not primarily behavioural.

Our work identified three different ways of classifying requirements (of all types):

(i) in terms of one (or more) of these viewpoints or projections of the computing system, the difference between them being the way in which the requirement is described.

(ii) in terms of the position within the design process where the requirement may be appropriately considered.

(iii) in terms of the ways in which a system can be evaluated were examined in order to see to what extent a particular requirement has been met.

The discussion of these three facets of NFRs (product, process and evaluation) led us to a more general view of requirements based on a categorization of each of these facets. (The aim in this work was to bring together what seems to be the most significant research ideas in the various facets identified above. In particular it drew heavily on the work of various other projects in which PDCS participants and their colleagues have been involved, namely the ISA (nee ANSA), ORDIT and METKIT projects, as well as PDCS.)

5. Fault Tolerance

The main thrust of activity on fault tolerance in our Project has been, and remains, aimed towards (i) producing more systematic methods for design, and (ii) helping a designer to choose fault-tolerant architectures, and individual mechanisms in them, for any individual new system, from the whole spectrum of those available, using the dependability specifications for the system as a guide to design decisions. Accordingly, in its first year the Project worked on a mix of classification work and ways to systematize parts of the design process. In its second year the Project has moved a step further, and dealt mostly with constructive design methods.
One aspect of our work has investigated fault assumptions and assumption coverage as a means of guiding two important parts of the design process. A classification of failure modes has been produced so as to help in the matching of fault-tolerant provisions in a system to the (formally defined) set of component failures that they are meant to tolerate. The notion has been introduced of "assumption coverage", and it has been shown how this can be used in the evaluation of a fault-tolerant design, and therefore in deciding design trade-offs, and in particular at which point adding redundancy to deal with the more unlikely failure modes becomes counterproductive.

Much of our work has dealt with the issue of designing fault tolerance in software. Methods and notations have been proposed that aim at controlling the complexity added to a design by the use of redundancy, while allowing the designer sufficient freedom in devising an effective scheme for each individual design.

A general framework and a corresponding program notation for software fault tolerance has been introduced. The purpose is to allow a programmer to use a wide variety of fault-tolerant structures, with a simple notation based on specifying separately the individual, independent characteristics of the desired structure. So, the spectrum of structures that can be defined includes, but is not limited to, all the schemes that have been proposed individually in the literature on software fault tolerance. The fault-tolerant modules so defined can be freely nested, so that the notation can be used at any level of functional decomposition in a design while abiding by good structuring principles.

The Project has also taken a closer look at the implementation of fault-tolerant program structures, and shown how basic fault-tolerant structures can be described in the design description language "BSM". This language, which is proposed as a means for designing concurrent applications with good characteristics of robustness and timing predictability, has been shown to be sufficient for expressing the typical structures of software fault tolerance without requiring semantic extensions. Detailed examples have been used to clarify how the features of the language are used.

Another strand of our work on fault tolerance has considered the problems that arise in the assembly of large application systems from components with heterogeneous provisions for fault tolerance. The assumption is that such situations cannot in principle be avoided, as provisions for fault tolerance at the application level are both needed and largely determined by the characteristics of the individual application modules. Rules for controlling the interaction among heterogeneous components have been proposed; one example of their use describes how components built using the conversation scheme can use server components implementing transactions.

Other work has applied the results of the well-established research field of system-level diagnosis to the systematic design of fault-tolerant architectures. It has been shown how theoretical results of system diagnosis can be used to devise new modular redundancy schemes (different from those diagnosis-based schemes), and existing results are surveyed. New techniques based on the diagnosability measure known as $t/(n-1)$-diagnosability have been developed, and examples devised showing their application to both hardware and software fault tolerance.
The fault tolerance approach has also been applied towards another facet of dependability, security. A detailed scheme has been presented in which access to data stored and transmitted in a distributed system is protected through consensus among distributed entities: a distributed, trusted server is obtained through the cooperation of several untrusted entities. This scheme has two important features: it preserves the administrative decentralization of the distributed system, and it has no single point of failure, with respect to either accidental faults or intrusions.

Between them, these various strands of work evidence the progress made towards developing methods for guiding and disciplining the systematic design of predictable fault tolerant computer systems. However further work is certainly needed, both in broadening the application of the investigation methods used to cover more general classes of problems, and to bring together the results of different lines of research. However, most of the results obtained to date are ready for use in tools in a design environment for predictably dependable systems, the ideal long-term goal of the PDCS project, and specifying such tools is one of the planned outcomes of this research.

6. Software Engineering Environments

The long term aim of the PDCS project is to develop an environment for the design and evaluation of predictably dependable computing systems. Achievement of this goal is a long way off, and there are many fundamental problems which have to be addressed and solved before such an environment can be produced, but we have made some progress on four sub-problems as indicated below.

Two involve pragmatic solutions to key sub-problems in developing dependable computing systems. Of these the first is concerned with the issues of developing highly reliable parallel programs for distributed systems, being particularly concerned with the adaptation of functional language-based approaches to such architectures. A prototype system (Paralex) has been constructed which supports the design of distributed applications that can be described using a data flow-like semantics. The interface presented to the programmer makes extensive use of graphics to define, edit, execute and debug applications. All of the necessary code for distributing the application across a network and replicating it to achieve the desired level of fault tolerance is automatically generated by the system.

The second sub-problem is concerned with the design and development of real-time systems and specifically addresses the environment that is evolving for the development of programs to run on the MARS system. The approach that has been developed covers the stepwise refinement of requirements, expressed in the form of real-time transactions, to task and protocol executions. It also includes a timing analysis and dependability evaluation of the still incomplete design. The testability of the evolving system is considered to be of essential concern.

The other two sub-problems concern an attempt to resolve two related long term research problems arising from the fact that the environment within which secure
applications are developed can itself have a large impact on the overall assurance attributable to the deliverable product. Our work on the first of these sub-problems concerns the development of high integrity environments for the production of secure application systems. To date the work has focused on the issue of verifying that an object-oriented design environment correctly implements an appropriate security/integrity policy.

The final sub-problem relates to the issue of determining such policies. Our work on the sub-problem has involved the production, so far mainly in outline, of an example of such an integrity policy, intended to provide appropriate controls over modification of the products of the software development process. To date the work has mainly concerned the sort of integrity and security policy of an environment that could be suited to the development of militarily secure systems, but many of the principles apply more widely to other classes of dependable system.

Although the work outlined above is at differing states of maturity all the strands contribute to the overall problem of producing an environment for the development and evaluation of predictably dependable systems, although little work has been done yet on incorporating evaluation concepts into the process. Future work will be concerned with drawing some of these strands together and addressing the evaluation issue.

7. Dependability Modelling and Evaluation

A key theme of the PDCS Project is that of modeling and evaluating a system's dependability. Recent work has been largely concentrated on four topics. The first is a general approach to reliability and availability, encompassing hardware and software, considering both stable reliability and reliability growth situations. The second focuses on availability of multiprocessor systems in stable reliability situations. The third deals with reliability evaluation of fault-tolerant software, allowing for reliability growth. The final involved processing experimental failure data using several different reliability growth models.

Work on X-ware (i.e. hardware and software) reliability and availability modeling addressed the problem of evaluating a system's reliability and availability with respect to the various classes of faults (physical and design, internal and external) which may affect the service delivered to its users. Reliability evaluations encompassing hardware and software are currently rare in spite of users' needs for evaluations expressed in terms of failures, independently of their sources, i.e. the various classes of faults. The causes of this situation have been analyzed and it has been shown that there is no theoretical impediment to performing such evaluations: classical reliability theory can be generalized in order to cover both hardware and software viewpoints. This has been demonstrated by a study, for the case of non-redundant systems, of system behavior up to (next) failure, and of sequences of failures.
The second topic concerns multiprocessor systems with general breakdowns and repairs. A performance and reliability model of a multiprocessor system, based on the $M/M/N$ queue with service interruptions, has been analyzed in the steady-state. The pattern of processor availability is governed by a rather general Markov process which allows multiple simultaneous breakdowns and/or repairs, and may depend on the job arrival and service processes. A novel solution method, called spectral expansion, was used to determine the joint distribution of the number of operative processors and the number of jobs in the system.

The third topic concerns reliability growth of fault-tolerant software. Fault-tolerant software approaches have given rise to a number of models. However, all of these models assume stable reliability, i.e. they do not consider reliability growth resulting from the progressive removal of design faults. Work on this topic has been aimed at addressing this problem. Stable reliability models have been derived for Recovery Blocks, N-Version Programming and N Self-Checking Programming. These models were then transformed into models that take into account reliability growth via a transformation approach based on the hyperexponential model. Analytical and numerical processing of the transformed models enabled identification of the influence of fault removal on the reliability of the each of the three fault-tolerant software schemes.

The fourth topic involved detailed analyses of a new set of software failure data. This data is of the failure behavior, over the last four years, of a particular single-user work station. The details recorded in this data collection exercise allowed the data to be subdivided into various subsets of inter-failure times. A sub-collection of these were chosen for more detailed analysis. Experience of applying reliability models in the past had shown that the relative predictive performance of the models depends entirely on the context. It had been found that there is no one model that performs well over all data sets. It had also been found that for some data sets all models applied are in error. In such cases two techniques for improving predictive accuracy have been shown to be beneficial: i) recalibrating the raw model predictions and ii) using the results of trend tests to apply the models. These two techniques may be used separately or in combination. The new analyses were mainly devoted to the first technique but the benefit to be gained by the application of the second technique was also shown. A number of reliability models together with the recalibration technique were applied to the failure data, and the performance of the resulting prediction systems was assessed.

In summary, as a result of all this reliability modelling work we pretty well understand how to obtain reasonably accurate reliability growth predictions in a wide class of circumstances. More importantly, we have techniques that allow us to know in a particular context whether the answers are ones that we should trust. All this has come about through our invention and validation of techniques such as u-plots, prequential likelihood, etc. for the analysis of predictive accuracy. These techniques have allowed us to develop a powerful and general new technique of dynamic model recalibration, which can improve the accuracy of models in particular circumstances, often with dramatic effect.
In addition, we have widened the scope of the usual models, enabling both reliability and availability of hardware and/or software systems to be evaluated, from knowledge of their components reliability. We are now able to model multi-component systems taking into account reliability growth of their components using the hyperexponential model and the associated transformation approach. As a consequence, important results have been derived for fault-tolerant architectures in the presence of reliability growth.

8. Analysis of Timeliness

The Project's research in this area has concentrated on some of the detailed problems which have to be solved if systems are to be built possessing predictable timeliness properties, and has centred on two main topics.

Whenever the execution of a real-time task must terminate before a specified time interval after its activation, i.e. before its deadline, knowledge about the maximum execution time of the task is essential. This knowledge is needed in time-triggered as well as in event-triggered architectures. In time-triggered architectures it is required at compile time as a necessary input to the off line scheduler while in event-triggered systems the on line scheduler must have access to this information at run time. There are a number of approaches possible to gain this knowledge: by observing the execution time distribution during the test phase, by analyzing the task structure in order to establish an upper execution time bound analytically or by observing the run-time behaviour of the task.

Work on the first topic involved experiments designed to investigate the feasibility of response time predictions. What we were looking for was a reasonable upper bound on the maximum response time of time critical tasks, considering all side effects of the operating system and hardware processes. In order to gain results which are representative for a large class of systems, we chose a very simple target architecture for our experiments. In the experiments several thousands of test runs of tasks were performed in order to measure the actual task execution times and the system overhead. The data so obtained were then compared with the analytically computed worst case execution times.

The results of these experiments were encouraging and discouraging at the same time. They were encouraging because it has been shown on realistic real-time control tasks that the analytical prediction of the maximum task execution time is not too far away from the observed "actual" maximum task execution time, disregarding system overheads. The results are discouraging, because even in the simple hardware architecture used the variability in the execution time, due to system overheads, is larger than the variability caused by the data dependencies. Based on these findings we are developing a new single board computer with two processors, one for the application software and one for the operating and communication system. We hope that this new hardware architecture will provide a superior base for the implementation of predictable real-time systems.
The second topic, which involved a study of the asymptotic optimality of the Go-Back-n Protocol in high speed data networks with small buffers, is concerned with the analysis of the tradeoff between dependability, resource usage, and real-time response of event-triggered communication systems.

The dependability of a communication network depends both on its hardware characteristics and on the protocol that controls it. The traffic rate that the network can support without incurring large delays is influenced not only by the speed of transmission, but also by the frequency with which packets have to be retransmitted due, for example, to buffer overflows.

As the transmission speeds of emerging data networks increase, the signal propagation delay, which does not scale, becomes a very important consideration in their design. For a congestion control protocol based on sliding windows, it was previously shown that the optimal window size grows linearly with the transmission speed, $\lambda$. Hence, the cost of memory used for intermediate buffers is a major factor. However, it was also shown that only $O(\sqrt{\lambda})$ packets are in the buffers at any one time, the rest being in transit between nodes. This suggested that it would be possible to use buffers that are much smaller than the window size, without increasing significantly the probability of overflow, i.e. without lowering significantly the reliability of transmission.

Our study quantified the above observation. With small buffers on the order of $O(\frac{1}{\sqrt{\lambda}})$, the ratio of the realized throughput to the ideal throughput was shown to approach unity as $\lambda$ increases. When properly sized, buffers overflow so rarely that even with a rudimentary (conversely, easily implemented) protocol like go-back-n, the loss in throughput due to retransmission is negligible.

The result was arrived at by obtaining an explicit characterization, for large $\lambda$, of the tail of the distribution of buffer occupancy in a closed network with window size buffers. That distribution was then used to estimate the overflow probability in the network with small buffers, and numerical and simulation results were obtained to confirm the predictions of the theory. The analytic theory involved is based on product-form networks but the essential tool is scaling. It is therefore our belief that the conclusions of this paper are more generally applicable and that the problem of establishing this is an appropriate subject for future research.

9. Statistical Assessment

Project work in this area has addressed the important issue of statistical assessment of the level of dependability achieved by a computing system. Two methods, namely statistical testing and fault injection, have been considered, aimed respectively at validating (i) software and (ii) fault tolerance.

Although they address distinct features of a dependable computing system, these methods correspond to two facets of a common testing effort. Indeed, although statistical testing deals classically with the activation domain of the software to be tested as the input domain, fault injection can be seen as a means for testing the fault tolerance
algorithms/mechanisms with respect to a particular class of inputs for the processing of which they have been specifically designed, i.e. the faults.

The methods for generating test inputs can be either deterministic or probabilistic. In deterministic testing, test data are predetermined by a selective choice according to the adopted criteria. With statistical (or random) testing, test inputs are selected according to a defined probability distribution on the input domain, and both the distribution and the number of input data are determined according to the adopted criteria.

Obviously, testing is always partial as it is not feasible to exercise a software with each possible sample from the whole input domain. A lot of test criteria have been proposed in the literature to act as guides in the determination of the test patterns. They relate to either the structure or the function of the software, leading respectively to structural testing and functional testing. Our work on testing has provided theoretical bases to quantify the quality of statistical testing with respect to a given criterion (or set of criteria), and has concentrated on the analysis of structural statistical testing. Experiments have been performed on four real life programs, and the results confirm the efficiency of structural statistical test inputs with respect to fault exposure.

Our work on fault injection has resulted in the development of a comprehensive framework suitable for identifying the main contribution of fault injection within the validation process. Fault injection also relies on a combination of deterministic and statistical selection of the injected faults among the injectable faults with respect to their location and timing characteristics. An experimental dependability evaluation method combining the fault tolerance coverage and the fault occurrence process has been developed. An original method has also been produced for facilitating the determination of a test sequence specifically aimed at removing fault tolerance design faults; the method focuses on the derivation of test criteria from the analysis of the fault tolerance mechanisms.

This work on testing and fault injection provides, we believe, a solid foundation for future research in the field of statistical assessment of dependability. The theoretical frameworks established will enable us to devise experiments well-suited (i) to improving the dependability of fault tolerant computing systems, and (ii) to quantifying the achieved dependability level.

The use of statistical assessment is necessary at the various phases of the design and implementation process. Depending on the particular phase, statistical assessment relies on different levels of abstraction for the models that describe the system (behavioral, structural models). The basic frameworks we have proposed are general enough to enable them to be applied at any of these abstraction levels. The first results obtained using them confirm their practical usefulness and their adequacy. We plan to investigate this issue further in the few remaining months of PDCS1, and we hope to be able to pursue the work in a future PDCS2 project.
10. Security Evaluation

Our work in this area has investigated a new direction of research in security evaluation by exploiting certain dualities between security and reliability, but also the practical aspects of current techniques of security evaluation, in fact via involvement in an extensive study of various PC-security products.

The work on 'operational measures' of computer security is distinguished from 'classical' security evaluation by striving for a probabilistic framework for security evaluation and for operational measures of security, as opposed to examining systems out of context of their actual environment. We have investigated similarities between reliability and security which could allow us ultimately to obtain measures of 'operational security' similar to those we have for reliability of systems. Very informally, these could involve expressions such as the rate of occurrence of security breaches (cf. rate of occurrence of failures in reliability), or the probability that a specified 'mission' can be accomplished without a security breach (cf. reliability function).

Our approach is based on the analogy between system failure and security breach. A number of other analogies have been shown to support this view. One has to distinguish between the user's viewpoint, as in reliability analysis, and the attacker's viewpoint. Indeed, the second view seems more appropriate for expressing security. There are still some open questions that have to be resolved to establish the feasibility of our approach. E.g., in a stochastic model of security we can define a security function, by analogy with the reliability function, to be \( R(c) = P(C > c) \), where \( C \) is the effort to the first successful breach. (This security function may well be given for a particular viewpoint.) We have to establish reasonable assumptions on \( R(c) \) as well as rules for the combination of attacks. To a large extent, these questions are of an empirical nature and could, at least in principle, be resolved by experiment and case study. The topic though new is very promising, and we hoped to continue our work on operational measures for security in a successor project to PDCS.

The classical security evaluation work in which we Project members have been involved was centred on an analysis of several commercial PC-security products. Products were examined by monitoring their execution using tools such as debuggers and logic analyzers. In this way, both the location of sensitive information (keys) and the detailed structure of the security mechanisms was reconstructed. Security evaluation is not a random (in the sense of undirected) search through memory or through sequences of commands. This search usually is performed at a level that is not the intended user interface. The study classified products according to the type of attack they can withstand. Attacks were classified according to the mode of access to the system, e.g. user interface - operating system - hardware.

On a technical level, the difference between good and bad security products can be found in the areas of cryptographic algorithms, key management, and systems integration. (There is no inherent reason why products making use of dedicated hardware should provide more security than pure software systems.) In our experience, all attempts at achieving security by denying information about the security system have
failed. In particular, the confidentiality of information stored in unprotected locations can only be provided by cryptographic means.

The effort for such an evaluation turned out not too prohibitive, and provided evidence that an experienced person working with the appropriate tools requires only several hours to a few days to establish how, or whether a given security system of the type studied can be penetrated. And although this study was not concerned with operational security, the observations made on the products and on the evaluation itself may help in formulating realistic assumptions on the models and distribution functions that are needed for the operational security evaluation techniques that we hope to develop.

11. Ultra-High Dependability

When work started in PDCS, we expected that the study we planned to make on ultra-high dependability would not have a high profile. There were several reasons for this, the most important being a recognition that this is a very difficult problem area. In fact, work has been quite intensive: as is often the case, a difficult (possibly insoluble) problem has thrown up some interesting ideas and questions which have wide-ranging implications for the whole project. It also sparked off some of the most interesting discussions and collaborations between sites.

One aspect of our work addressed the limits of the various means available for validation: reliability growth models, testing with stable reliability, structural dependability modelling, as well as more informal arguments based on good engineering practice. It showed that no one of these means alone allows the validation of extreme dependability requirements, and has started to investigate how all these kinds of information can be taken into account together.

Another study has concerned how to augment these techniques by explicitly taking account of the development process in the quantitative assessment of the reliability of its product. This requires the development of methods of combining information about system dependability coming from disparate sources, most particularly from the development process; the expected limits of such methods have been analyzed.

A third study has analyzed the advantages and limits of formal approaches to software development for achieving ultra-high dependability of safety-critical computing systems. Among the issues addressed were such questions as: what is a formal specification? what is correctness? what kind of certainty comes from proof? from testing?

These three studies are closely related, in that they addressed different aspects of essentially the same problem: how to acquire confidence in a system that has extremely high dependability requirements. The first two considered the problem of the evaluation of the effects upon dependability of remaining faults on the assumption that deterministic formal techniques for the verification of software usually cannot guarantee
that it is fault-free when judged against engineering requirements. The third considered what can be concluded from this kind of formal analysis.

A fourth study addressed the important question of requirements analysis for safety-critical systems. A general framework was developed for the formal specification and verification of the critical requirements in the development of safety-critical systems. The framework was illustrated via an example based on a computer-controlled railway level crossing.

In summary, we believe that this work represents an excellent foundation for future work within the PDCS project. The work on validation gives a good understanding of the scope of the different approaches to achieving confidence that a particular system has achieved its required dependability. In particular, we now have a better awareness of where the limits of these techniques lie and are able to see ways in which these limits might be pushed back.

One of the special characteristics of the PDCS project has been its fostering of an interaction between those of us working on methods to achieve dependability, and those working on methods of evaluation. We believe that in this work on the very difficult problems of ultra-high dependability this collaboration between the two viewpoints has been particularly successful. It is our intention to work on these problems in the remaining months of PDCS1, and we hope that a future PDCS2 project will allow the work to continue further.

Acknowledgements

The named authors of this paper are not, of course, the sole creators of the words it contains. We are grateful to our many colleagues in the Action for their contributions to the project reports which have provided raw material for our selection.
Appendix 1: Contents of the second year report

Volume 1

Chapter 1: Dependability Concepts

Summary and Conclusions: A. Burns, J.E. Dobson, N.E. Fenton, J.C. Laprie, J.A. McDermid, B. Randell

Validating Software Measures - N.E. Fenton (City University), B.A. Kitchenham (National Computing Centre, Manchester)

Chapter 2: Effects Of Operational Environments On Dependability

Summary and Conclusions: D. Wright, B. Littlewood

Part 1: Incorporating Explanatory Variables in Software Reliability Models - D. Wright (City University)

Chapter 3: Specification And Design For Timeliness

Summary and Conclusions: A. Burns, G. Fohler, H. Kopetz

Part 1: Real-Time Objects and Temporal Uncertainties in Distributed Computer Systems - H. Kopetz (Technische Universität Wien), K.H. Kim (University of California)

Part 2: Incorporating Flexibility into Offline Scheduling for Hard Real-Time Systems - A. Burns (University of York), G. Fohler (Technische Universität Wien)

Chapter 4: Dependability Requirements

Summary and Conclusions: J.E. Dobson, J.A. McDermid

Part 1: On "Non-Functional" Requirements - J.E. Dobson (University of Newcastle upon Tyne), J.A. McDermid (University of York)

Chapter 5: Fault Tolerance


Part 1: Fault Assumptions and Assumption Coverage - D. Powell (LAAS-CNRS)

Part 2: A General Framework for Software Fault Tolerance - C. Liu (University of Newcastle upon Tyne)
Part 3: Structuring Software Fault Tolerance with BSM - A. Bondavalli (CNUCE-CNR), L. Simoncini (Universita di Pisa)

Part 4: Flexible Schemes for Application-Level Fault Tolerance - Lorenzo Strigini, F. Di Giandomenico (IEI-CNR)

Part 5: Fault Tolerance Based on System Diagnosis Techniques - J. Xu (University of Newcastle upon Tyne)

Part 6: Intrusion Tolerance in Distributed Computing Systems - Y. Deswarte, L. Blain, J.C. Fabre (LAAS-CNRS)

VOLUME 2

Chapter 1: Software Engineering Environments


Part 1: Paralex: An Environment for Reliable Parallel Programming in Distributed Systems - O. Babaoglu, L. Alivisi, A. Amoroso, R. Davoli (Università di Bologna)


Part 3: Towards an Object Oriented Development Environment for Secure Applications - E.S. Hocking, J.A. McDermid (University of York)

Chapter 2: Dependability Modelling And Evaluation


Part 1: X-Ware Dependability Modeling and Evaluation - J.C. Laprie, K. Kanoun (LAAS-CNRS)

Part 2: Dependability Evaluation of Multiprocessor Systems Subject to Breakdowns and Repair - R. Chakka, I. Mitrani (University of Newcastle upon Tyne)


Part 4: Analyses of Software Failure Data - S. Brocklehurst, B. Littlewood, P. Mellor, A. Tanner (City University), K. Kanoun, S. Metge (LAAS-CNRS)
Chapter 3: Analysis of Timeliness

Summary and Conclusions - H. Kopetz, I. Mitrani, P. Puschner, A. Vrhoticky


Part 2: Asymptotic Optimality of the Go-back-n Protocol in High Speed Data Networks with Small Buffers - D. Mitra (AT&T Bell Labs), I. Mitrani (University of Newcastle upon Tyne)

VOLUME 3

Chapter 1: Statistical Assessment

Summary and Conclusions - J. Arlat, Y. Crouzet, J.C. Laprie, P. Thévenod-Fosse, H. Waeselynck

Part 1: On the Efficiency of Statistical Testing with respect to Software Structural Test Criteria - P. Thévenod-Fosse (LAAS-CNRS)


Chapter 2: Security Evaluation


Towards Operational Measures for Computer Security - S. Brocklehurst, N.E. Fenton, B. Littlewood, P. Mellor, S. Page, D. Wright (City University) J.E. Dobson (University of Newcastle upon Tyne), D. Gollman (Universität Karlsruhe), J.A. McDermid (University of York)

Chapter 3: Ultra-high Dependability


Part 1: Validating Ultra-High Dependability for Software Based Systems - B. Littlewood (City University), L. Strigini (IEI-CNR)

Part 3: Advantages and Limits of Formal Approaches for Ultra-High Dependability - M.C. Gaudel (LRI - Université de Paris-Sud et CNRS)

Appendix 2: Project Publications in Refereed Technical Conferences and Journals


