Communications are Everything: A Design Methodology for Fault-Tolerant Concurrent Systems

A.M. Tyrrell
Department of Electronics
University of York, Heslington, York, Y01 5DD, Email: amt@ohm.york.ac.uk

Abstract
Limiting the extent of error propagation when faults occur and localising the subsequent error recovery are crucial elements in the design of fault tolerant parallel processing systems. Both activities are made easier if the designer associates fault tolerance mechanisms with the underlying communications of the system. With this in mind, this paper has investigated the design of such systems, which enforces a design concentrating on the modelling and analysis of interprocess communications providing a better match between the needs of the fault-tolerant mechanisms and the communication structures themselves.

Keywords
Fault-tolerance, concurrent systems, communications, software design.

1 INTRODUCTION

A distributed processing system, comprising a set of discrete processing units, offers the user not only the prospect of increased efficiency and throughput through parallelism, but its inherent redundancy might also be exploited to enhance reliability. To do so requires a properly designed fault tolerance infrastructure which maintains the integrity of the system under fault conditions, in particular communications. This paper describes a design methods which concentrates on the communications within the system, which facilitates the design, placement and implementation of fault-tolerant software mechanisms across a parallel system to ensure safe operations in the presence of faults.

Fault tolerance is often incorporated into a design as a ruggedisation process to protect a process or set of processes regarded as critical to safe system operation (Lee and Anderson 1991). The fault tolerance mechanisms are required to recognise faults by the errors they cause and to prevent error migration from the faulty process to elsewhere in the system, so that error recovery is localised. The extent of the error recovery operation can be limited if the communications structure in the system can be analysed accurately, and a boundary can be identified within the state-space of the distributed system across which error propagation by interprocess communication is impossible; it must include all processes which interact with the function being protected and exclude all processes that do not interact with it. In other words, the state-space of the system has to be partitioned into a hierarchy of atomic actions (Jalote and Campbell 1986). It is then possible to design a distributed error detection and recovery mechanism around the atomic action which ensures that all the processes affected by the fault co-operate in recovery. This localisation of fault tolerance simplifies the design and can help to meet timing constraints in real-time systems (Anderson and Knight 1983).

The design described in this paper concentrates on the communications mechanisms within an application, and within the fault tolerance mechanisms themselves. The design shows how different communication structures help not only in the design of the particular application itself, but more importantly in the design of the fault-tolerant mechanisms protecting the system against faults latent in the system.

2 ATOMIC ACTIONS AND FAULT-TOLERANCE

Firstly, let us consider the crucial role communications play in the operation of fault-tolerant mechanisms in a parallel processing environment. To an external observer the activity of a process is defined by its sequence of external interactions; any internal actions (of which there may be many) can not affect the external observer, at least until the next external interaction. This allows the concept of an atomic action to be derived: the activity of a set of processes is defined as an atomic action if there are no interactions between that set of processes and the rest of the system for the duration of that activity. The extension to hierarchically nested atomic actions is straightforward. These concepts are well-known in distributed transaction processing (Mancini and Shrivastava 1988) from which field many other attributes of atomic actions, such as serialisability, failure atomicity and permanence of effect can be defined.

The process of identifying the atomic actions within a parallel system design brings into clear focus the structure of interprocess interactions and thus the route by which errors might propagate under fault conditions — an obviously crucial aspect in the detection and implementation of the fault tolerant mechanism. All common mechanisms for providing fault tolerance in parallel systems, such as forward error recovery (Randell 1975), N-version programming (Avizienis 1985), conversations (Randell 1975), consensus recovery blocks (Scott et al. 1987) and distributed recovery blocks (Kim and Welch 1989), have to cope with error confinement and achieve this by imposing logic structures 'around' atomic actions.

A generalised fault tolerant mechanism could be considered as a co-ordinated set of recoverable blocks, with one recoverable block in each interacting process, allowing distributed error detection and recovery. The mechanism is bounded by a set of start states (*entry line*), a set of finish states (*exit line*) and two side walls which completely enclose the set of interacting processes which are party to the mechanism, and across which interprocess interactions are prohibited. The structure is indicated diagrammatically in Figure 1. Note that it is the communication pattern that defines the side walls, processes which are interacting are within the side walls (processes R, S and T), processes which do not interact are outside the side walls (processes P and Q).

Two types of communications are illustrated in Figure 1; the lines between the 'recoverable processes' represent the *application* interactions, and are of a consequence of data requirements between the parallel processes. It is these interactions that will define where atomic action exist within the system structure, and thus where fault-tolerant mechanisms should be placed. The second type of communications are those forced upon the application by the fault-tolerant mechanisms. These will typically consist of exchanging data values for voting and/or for comparison, of passing reconfiguration information and signals around the system, and for the recovery of the parallel processes within the fault-tolerant mechanism. This second class of communication would not be present in non fault-tolerant systems, and in many respects should be more secure than the 'normal' application communications.

The entry line defines the start of the atomic action and consists of a co-ordinated set of recovery points for the participating processes. These processes may enter the atomic action asynchronously. The exit line comprises a co-ordinated set of acceptability tests, or voting procedures. Only if all participating processes pass their respective acceptability tests (or the voting procedures are successful) is the mechanism deemed successful and all processes exit, in synchronism, from the action. If any acceptability test is failed, recovery is initiated and processing "passed" to another set of recoverable processes, or set of actions. Thus all processes in the atomic action co-operate in error detection. Note how both synchronous and asynchronous communication structures are present in these mechanisms.

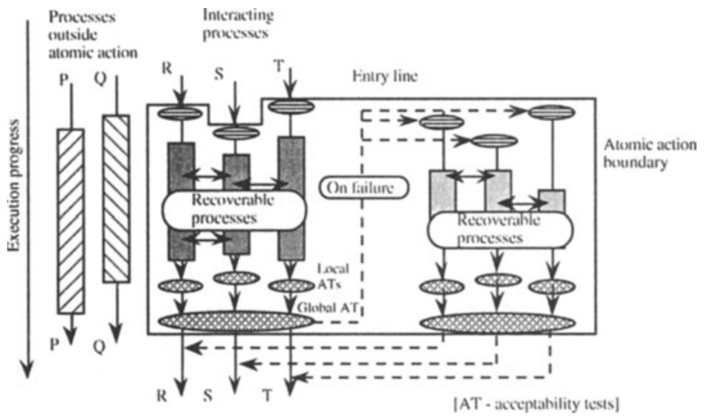


Figure 1. The structure of a fault tolerant mechanism involving processes R, S, and T.

Any attempt to incorporate an entry line and an exit line at arbitrary locations in a concurrent system is unlikely to lead to a properly formed recovery mechanism. It is necessary to identify a boundary within the state space of the complete set of processes across which error propagation by communication is prevented (Tyrrell and Carpenter 1995). Clearly, this boundary will be the boundary of an atomic action, since such a boundary, of necessity, prohibits the passing of information to any process not involved in the atomic action and similarly embraces all interacting processes within the atomic action. Recovery mechanisms can be nested systematically in the same hierarchical fashion as atomic actions. If this duality is not imposed, then should the system attempt to backtrack and recover in response to a fault, progressive collapse by the domino effect (Randell 1975) can occur.

3 FAULT MODEL

It is important at this stage to say a little about the types of faults that can be expected in the systems that are being considered. The fault model for these system comprises of both software and hardware faults.

Hardware Faults :

- dead processor (due to failure of processor or support chips),
- dead interprocessor communication (due to failure of communication hardware),
- erroneous interprocess communication (due to transient fault in processor or communication hardware).

Software Faults :

- differential mode faults (ie. software versions fail independently of each other),
- common mode faults (ie. software versions fail in same manner under the same conditions),
- faults due to difficulty factor (ie. versions fail in different ways under the same system conditions).

While more subtle and complete fault models have been suggested, this fault model provides sufficient ability to give a good idea of the effectiveness of the fault-tolerant mechanisms under consideration.

4 COMMUNICATIONS MODEL

A common communications model used in many fault tolerant systems is that of communicating sequential processes (CSP). This model provides a synchronous non-buffering communications procedure only. While this allows analysis of communications structures, and a effective implementation environment, eg transputers, there are some limitations to this model when used for fault-tolerant mechanism design and implementation. This form of communication is useful for the description of communications that are required between processes that are being forced into synchronous operation at points through their non-synchronous (asynchronous) execution. Problems do occur when implementing and analysing such system designs, when time-outs are introduced to prevent these synchronous communications from allowing a faulty process to stop non-faulty processes.

A more comprehensive suite of communication mechanisms are required if fault-tolerant mechanisms are to be really useful in real applications. Such a communications model has been described by Simpson (Simpson 1994a). This communications model will be used here to design fault-tolerant mechanisms and show how they would be implemented with such a model.

The communications model can be broadly categorised into one of four regions (Simpson 1994b), this is illustrated in Figure 2.

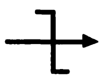
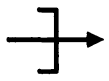
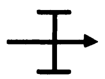
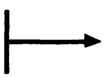
	Destructive Reading *Reader can be held up*	Non Destructive Reading *Reader cannot be held up*
Destructive Writing *Writer cannot be held up*	 SIGNAL *Event Data*	 POOL *Reference Data*
Non Destructive Writing *Writer can be held up*	 CHANNEL *Message Data*	 CONSTANT *Configuration Data*

Figure 2 Communication Model.

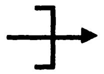
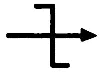
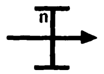
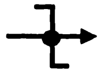
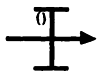
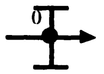
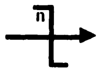
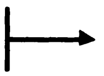
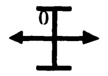
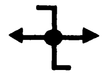
Interaction Function	Symbol	Writer Can Be Held Up	Reader Can Be Held Up
POOL		N	N
SIGNAL		N	Y
CHANNEL/BOUNDED BUFFER		Y	Y
STIMULUS/INTERRUPT		N	Y
RENDEZVOUS		Y	Y
HANDSHAKE		Y	Y
OVERWRITING BUFFER		N	Y
CONSTANT		-	N
REMOTE FUNCTION CALL		Y	Y
REMOTE THREAD INVOCATION		Y	Y

Figure 3 Enhanced Communication Model.

In (Simpson 1994b) these communication models are described as follows: *Pools* allow reference data to be passed from one function to another. This data is retained within the pool where it can be consulted at any time by the reader and refreshed at any time by the writer. *Signals* allow event data to be passed from one function to another. This data can be overwritten at any time by the writer, but can only be actioned once by the reader. The signal is important in real-time systems as it avoids back propagation of temporal interaction effects. *Channels* allow message data to be passed from one function to another. It is normal for channels to have a capacity of more than one, when they become synonymous with a bounded buffer. The message in a channel cannot be lost. *Constants* can be considered as configuration data, and provide a write-once capability.

Note in this model the categories are with respect to destructive reading and writing. Within this simple model, "standard" message passing can be fixed, as can a form of shared data space, asynchronous data, and even global data.

This model can be further enhanced to incorporate system characteristics such as buffer size (shown as n or 0), uni- and bi-directionality (shown by the arrows on lines) and non-data passing (ie stimulus only) indicated by a blob. This enhanced model (Simpson 1994b) is shown in Figure 3.

These enhancements give what is thought as a full model of communication procedures required to describe computer systems. It is noted in (Simpson 1994a) that *signal* and *pool* variants are the most useful for real-time applications. These are difficult to model in synchronous-only models, however this model provides the dynamic characteristics required for these models. We will now go on to show how this communications model can be used to design fault-tolerant mechanisms in a parallel environment.

5 FAULT-TOLERANT DESIGN FOR CONCURRENT SYSTEMS

The use of atomic actions enables many of the problems associated with introducing fault-tolerance into distributed/parallel system to be solved. One of the major problems with these ideas is that in order to identify processes that may be considered atomic actions, the dynamics of the processes and thus the state space of the system must be modelled. The author has successfully achieved such analysis by using Petri-net and GMB graphical methods, and CSP mathematical methods (Carpenter and Tyrrell 1989, Tyrrell and Holding 1986, Tyrrell and Carpenter 1995).

For the introduction of fault-tolerant mechanisms to aid the system designer, there should be provided a set of *framework processes* within which the application program will sit. The structure of these framework processes should be of no concern to the application designer, the only application specifics in its incorporation into the design should be the design of the error detection mechanisms (whether hardware or software, this will always be application dependent), and in the actual placement of the framework process across the distributed/parallel system.

Once the atomic action boundaries have been identified, the chosen framework process (such as, forward error recovery, backward error recovery, and error masking) can then be placed around these safety critical application processes. We will now look at one of these mechanisms, and see how the communications model helps in its design (and in a final implementation).

Enhanced Distributed Recovery Blocks.

The mechanism used is based on distributed recovery blocks (Kim and Welch 1989). It is argued that distributed recovery blocks (DRB) are well suited for real-time control applications since:

- DRB require code versions of graded complexity; a requirement which should easily be satisfied by the plethora of new and classical control theories which are in existence,
- DRB offers distributed operation over a number of redundant processing nodes,
- In the event of a fault DRB dynamically reconfigures the operation of these nodes in order to obtain the maximum possible performance from the hardware available,
- In the event of faults DRB will fail gracefully, always using the highest graded code version available to it,
- DRB relies on acceptance tests, rather than voting, to judge the correctness of results; this is important as voting between alternative control algorithms can be unreliable due to their tendency to produce correct, but different results,

- DRB is proposed as a uniform way of dealing with hardware and software faults; it obviates the need to identify the origin of a fault, which is a costly overhead in terms of time, and has been a major difficulty in real-time computing designs.

DRB are based on the standard method of recovery blocks, Figure 4. The enhancements incorporated within DRB include the concurrent execution of the try blocks over a distributed network of processing nodes and the dynamic reconfiguration of nodal operations in the event of a fault. The systems proposed in this paper takes the basic DRB and introduces extra acceptance tests to reduce the chances of Byzantine type errors and is termed an Enhanced DRB (EDRB), Figure 5. The acceptance test in the EDRB scheme are carried out concurrently on N different nodes. In addition the local database of previous data which is maintained on each node will be exactly the same. The DRB maintains separate databases on each node these are regularly exchanged and compared, thus guaranteeing that they are the same and eliminating the need for any form of roll-back recovery in the event of a detected error. EDRB performs a vote at the start of every iteration to ensure each node is operating with exactly the same data.

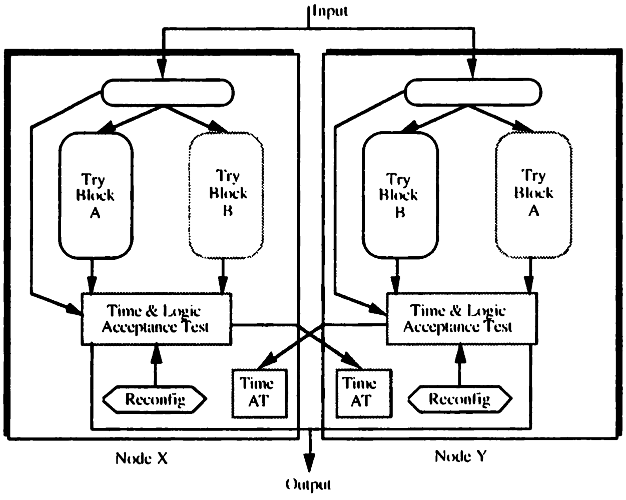


Figure 4 Distributed Recovery Block.

6 COMMUNICATION FAILURES

In an earlier paper on the EDRB (Elphick et al. 1993), a design was reported which in addition to designing the EDRB around real-time applications a number of features were added to the design to help cope with communication failures. When implemented with a CSP model, link procedures were used to detect communication failures (or time-outs), ensuring that other non-failed nodes continued to operate correctly. Reinitialisation procedures were also used to reset failed nodes and allow them to be re-included on the next iteration of the loop (assuming non-permanent hardware faults). The calculation of these time-outs were non-trivial and prone to errors.

These link procedures where very much a consequence of the CSP, synchronous model used. Here it is shown how this new communications model gives a more general solution, allowing particular interactions between parallel processes to be more closely modelled by specific communication structures.

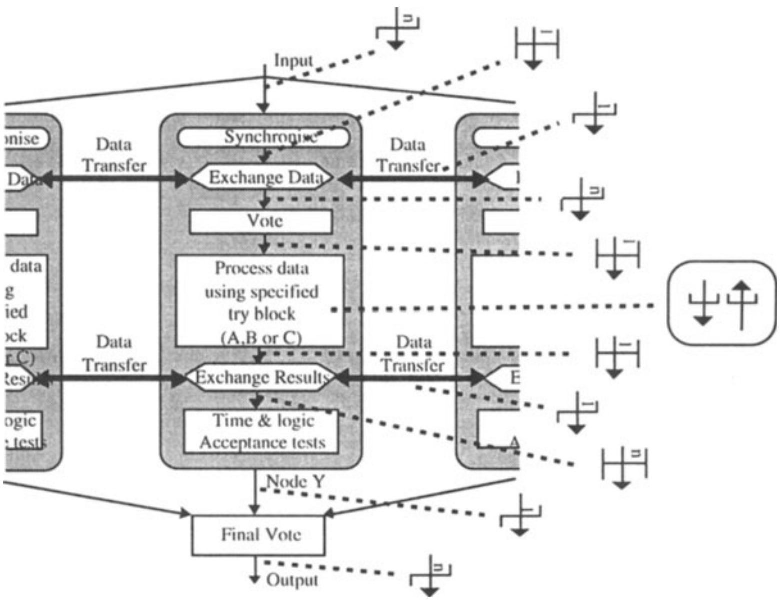


Figure 5 Enhanced Distributed Recovery Block.

Generally, it can be seen in this design that the communications consist of a number of signal and overwriting buffers, and channel or bounded buffer interactions. During the application processing itself, some reference data may be read/written — illustrated by the pool interactions. Indeed some timing information is likely within this process itself, and can be easily specified using this design method. A simple control loop is shown in Figure 6 (Simpson 1994a) illustrating this design method. More detail of this is given in (Simpson 1994a). The choice of whether the interaction should be a signal or a channel is dependant on if the writer should (can) be held up by the reader or not. This is important, for example, when interaction with processes external to the atomic action (ie at the input and output) so that the processes time constraints do not affect the external environment (eg this could be the controlled process). Another example for the use of signals is when the parallel processes are exchanging data for voting and comparison; here we would not wish the writing process to be held up. In certain cases however, it is important that both writer and reader are held up, for example, when the input data is read in and the processing cycle of each process is synchronised.

It appears that by using this new, more general communications model, the system design is better suited to the functions required of it by the particular interactions between the parallel processes. The richer set of communication primitives in this model enables explicit communication structures to be built for specific jobs within the system; in particular in this application, specific for the EDRB. These explicit communication structures force the designer of the system to consider the most applicable structures for the particular interactions. This should produce a design closer to the application, and hopefully a design that is less likely to perform incorrectly in the final implementation. The implementation of such a system should also be easier than an equivalent one using just synchronous communications; Using this richer set of communication structures, many of the timing problems associated with the purely synchronous design disappear allowing a simpler design to be arrived at. Obviously, one has to pay for such an improvement! Many of the problems associated with the purely synchronous design methods are removed by the more "complex" set of mechanisms provided in this new communication set. One could consider that the problems have now been removed to the hardware mechanisms controlling the communications. This assumes that hardware mechanisms are available to implement the different communication structures. It is mentioned in (Simpson 1994b) that chip support for these communication primitives is being designed in the form of a kernel executive chip and a comms executive chip.

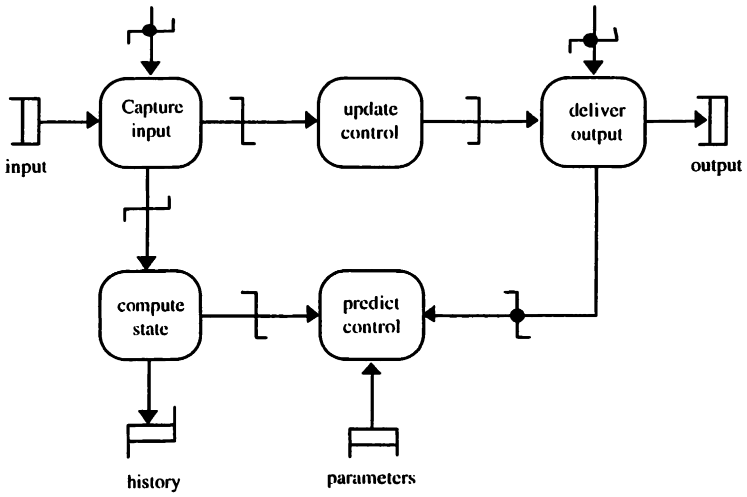


Figure 6 Example Control Process.

7 CONCLUSIONS

This paper has proposed that communications are crucial to the design, and implementation of fault-tolerant mechanisms applied to parallel processing systems. It has shown how a more general model of communications, than that of synchronous communications, can provide better mapping from what is required in fault-tolerant mechanisms. This has been illustrated by a design of a particular fault-tolerant mechanism, but as with the concept of atomic actions, these are general conclusions and should be applied to all fault-tolerant mechanisms designed for operation within a parallel environment.

It has been shown in previous papers that atomic actions should form the basis for fault tolerant mechanisms in a parallel environment; this paper shows how they can be designed and implemented in a systematic, proper fashion. Work is continuing to improve this design method for fault-tolerant mechanisms, as is an implementation of these communication structures by others.

It was proposed in a previous paper (Tyrrell 1994) that a set of design procedures for fault-tolerant distributed/parallel systems could be as follows:

- design a set of application processes,

- model these using an appropriate state space method,
- identify the safety critical functions of the system,
- identify the atomic actions associated with these safety critical processes,
- place the appropriate framework process(es) around these atomic actions,
- design error detection mechanisms for the application in question.

The ideas proposed in this paper would support these design procedures, and enhance them by the introduction of a rich set of communication primitives allowing the mappings from one stage of the design to the next to be achieved easily and naturally.

8 ACKNOWLEDGEMENT

The author would like to thank Geof Carpenter and Hugo Simpson for their ideas on some of the work related in this paper.

9 REFERENCES

Anderson, T. and Knight, J.C. (1983) A framework for software fault tolerance in real-time systems. IEEE Transactions on Software Engineering, **9**, **12**, 355-364.

Avizienis, A. (1985) The N-version approach to fault-tolerant software, IEEE Transactions on Software Engineering, **11**, **12**, 1491-1501.

Carpenter, G.F. and Tyrrell, A.M. (1989) The use of GMB in the design of robust software for distributed systems. Software Engineering Journal, **4**, 268-282.

Elphick, J.R. Patton, R.J. and Tyrrell, A.M. (1993) Enhanced Distributed Recovery Blocks: A Unified Approach for the Design of Safety-Critical Distributed Systems. IEE Colloquium on Safety Critical Distributed Systems, IEE London, Digest No: 1993/189.

Jalote, P. and Campbell, R.H. (1986) Atomic actions for fault tolerance using CSP. IEEE Transactions on Software Engineering, **12**, **1**, 59-68.

Kim, K.H. and Welch, H.O. (1989) Distributed execution of recovery blocks: an approach for uniform treatment of hardware and software faults in real-time applications. IEEE Transactions on Computing, **38**, **5**, 626-636.

Lee, P.A. and Anderson, T. (1991) Fault Tolerance: Principles and Practice. Springer Verlag.

Mancini, L.V. and Shrivastava, S.K. (1988) Replication within atomic actions and conversations: a case study in fault-tolerance duality. FTCS-19, Chicago, 454-461.

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

Scott, R.K. Gault, J.W. and McAllister, D.F. (1987) Fault-tolerant software reliability modelling. IEEE Transactions on Software Engineering, **13**, **5**, 583-592.

Simpson, H.R. (1994a) Temporal Aspects of Real-Time System Design. IEE Colloquium on Methods and Techniques for Real-Time System Development, IEE Press.

Simpson, H.R. (1994b) Architecture for Computer Based Systems. Proceedings of the 1994 Tutorial and Workshop on Systems Engineering of Computer-Based Systems, Stockholm, 70-82.

Tyrrell, A.M. and Holding, D.J. (1986) Design of reliable software in distributed systems using the conversation scheme. IEEE Transactions on Software Engineering, **12**, **7**, 921-928.

Tyrrell, A.M. (1994) The Design of Fault Tolerant, High-Performance Control Systems. IEE Colloquium on High-Performance Computing for Advanced Control, IEE London, Digest No: 1994/241.

Tyrrell, A.M. and Carpenter, G.F. (1995) CSP Methods for Identifying Atomic Actions in the Design of Fault Tolerant Concurrent Systems. IEEE Transactions on Software Engineering, **21**, **7** 629-639.

10 BIOGRAPHY

Dr Tyrrell received a 1st class honours degree in 1982 and a PhD in 1985, both in Electrical and Electronic Engineering. He joined the Electronics Department at York University in April 1990, and was promoted to Senior Lecturer in 1995. Previous to that he was a Senior Lecturer at Coventry Polytechnic. Between August 1987 and August 1988 he was visiting research fellow at Ecole Polytechnic Lausanne Switzerland. His main research interests are in the design of parallel systems, fault tolerant design, software for distributed systems, simulation using parallel computers and real-time systems. In the last five years he has published over 60 papers in these areas, and has attracted funds in excess of £500,000.