Abstract:

Since October 1978, an SRC sponsored project at the Computing Laboratory of the University of Newcastle upon Tyne has been investigating the reliability and integrity of distributed computing systems. This report documents the progress of this project up to September 1980 and lists all of the formal and informal papers that it has published.
1. Introduction

Of the two complementary approaches to achieving highly reliable systems, namely fault tolerance and fault avoidance, our project concentrates on the former. It is a sequel to a project which undertook one of the first serious investigations of the concept of design fault tolerance. The definitions introduced by this project are now becoming increasingly widely accepted: for example, the concepts of failure, fault and error, and the classification of fault tolerance strategies into those for error detection, damage assessment, fault treatment and error recovery. The current project, like its predecessor with the introduction of the recovery block scheme[1], is concentrating on system structuring techniques aimed at facilitating the choice of suitable fault tolerance strategies and their incorporation into computing systems. In particular we are making extensive investigations of error recovery strategies.

Although distributed computing systems have some reliability advantages deriving from the availability of multiple hardware resources, they can suffer from reliability problems which do not occur in "conventional" centralised computing systems. These problems arise from the existence of concurrent, yet inter-dependent, activities in a distributed system. For example, one part of the system may be taking decisions and performing actions in ignorance of the fact that these decisions have been based on copies of data which are already out of date, or even that some other part of the system has already determined to be erroneous. Such problems may exist in any type of distributed system, but can be considerably exacerbated by the telecommunications difficulties and delays incurred in geographically distributed systems.

The specific problem that the Newcastle project is concentrating on is that of designing distributed computing systems which can cope effectively with situations in which it is belatedly detected that erroneous data has been allowed to spread from computer to computer in the system. Ideally, of course, such spreading would be prevented. In practice, such prevention can never be guaranteed, if for no other reason than the fact that it is impossible for a system (or a system component) to check that the inputs it receives are indeed correct with respect to the aims of its environment rather than just consistent with data received previously, and valid with respect to pre-specified criteria.

The current project is a natural successor to the first reliability
project at Newcastle. Taken together they show a consistent concentra-
tion on the problems of tolerating severe faults. Initially these were
considered just in simple systems (i.e. an isolated sequential process),
but gradually more complexity was assumed — first of all the use of con-
ventional I/O devices, then parallel activity under centralised control,
and now distributed systems are being considered. An alternative (and
we believe less preferable) sequence of research activity which some
other groups have followed is that characterised by concentration from
the outset on highly complex systems, whilst making initial optimistic
assumptions about the difficulties that different classes of fault can
cause. In other words, we argue that complexity is a more suitable
"add-on feature" for system designers to implement than reliability!

The initial statement of the aims of the project indicated that the
approach to be followed would be centred on

1. the development of a 'system-level design notation' which provides
   means of expressing policies and strategies relating to the overall
   reliability of the distributed system, and the integrity of the
global data that the system is expected to maintain, and

2. the design of efficient decentralised (and in fact highly parallel)
   mechanisms which can embody such policies and strategies.

The current project, like its predecessor, is endeavouring to
develop concepts and techniques of general applicability, rather than
concentrate on the particular problems of individual specialised appli-
cations and environments, or of a particular manufacturer's hardware and
software. Nevertheless, some basis had to be chosen for experimental
prototypes to be constructed by the project to complement the more
theoretical activities by assisting in the evaluation of the practical-
ity and the limitations of various proposed strategies and mechanisms.
This basis is being provided by suitable augmentation of existing
hardware resources (two PDP11/45s acquired by the original reliability
project) so as to form a small network in which each node runs a version
of the UNIX operating system. Our choice of using UNIX virtually as a
basic system component means that our research tends to be in terms of
data base system strategies and computer network protocols, rather than
processor architecture and low-level programming concepts. However we
do not believe this implies any essential loss of generality, since it
can be argued that the system structuring concepts we have identified
and explored are in general applicable at all levels of a system.

The concept of multi-level systems, in fact, forms one of the cen-
tral themes of our work, and is discussed in some detail in section 3 of
this document. Subsequent sections deal with the other major technical
areas on which we have concentrated, and with such matters as project
resources and interactions with other groups. Complete lists of the
formal and informal publications issued by the project are also given.

2. Staff

In order to set the scene for the discussions that follow it is
beneficial to introduce the names of the staff involved. Since this
project began two research associates, Tom Anderson and Santosh Shrivastava, have been appointed to lectureships in the Computing Laboratory, but happily continue to be associated with the project. Flaviu Cristian and Graham Wood, recent recipients of Ph.Ds from Grenoble and Edinburgh respectively, have been appointed in their place, and constitute with Pete Lee and Eike Best the project's full-time staff. Keith Heron, employed half-time on the project until September 1979, has now resumed his full-time position in the Computing Laboratory, although still provides valuable assistance to the project.

During the current academic year we have benefited from the presence of Maurice Jegado, an IRIA sponsored visitor to the project. A further visitor from France, Alain Chesnais, a Ph.D. student at the Universite de Paris-Sud, is also spending 3 months with the project this summer.

Last, and by no means least, mention should be made of Fabio Panziari whose RSRE sponsored project has since June 1979 enabled him to work closely with the main reliability project.

3. Fault Tolerance in Multi-level Systems

It is now widely recognised that the complexity inherent in the majority of software systems is the major cause of their unreliable operation. In consequence, much research effort has been expended in the search for methods of structuring a system so that its complexity can be mastered. In particular, approaches involving structuring a system into a hierarchy of interfaces (or levels of abstraction) have received considerable attention. Basically, these approaches involve extending a given (possibly hardware) interface through the provision of programs (or object managers) which implement new abstract objects and provide operations to manipulate these objects. By successive extensions, an interface can be provided to match the characteristics of the intended application. Support for this interface can be constructed in a structured manner by small, well-defined modules. This may be contrasted with the more usual approach involving the construction of an essentially monolithic piece of software to provide the same features.

Structuring a system in this manner will be beneficial to the reliability of the overall system, since it can be expected that fewer faults will be present in the implementation of small, well-defined modules. Even so, there can be no real guarantee that any module is completely free from design faults, and our particular interest in multi-level systems arose in the application of fault tolerance in such systems.

Our early investigations addressed the provision of backward error recovery in multi-level systems, since this form of recovery has the attractive property of being able to provide automatic recovery even from unanticipated design faults. Basically, our investigations were concerned with: (i) the primitive features which have to be provided on an interface in order to support backward error recovery for programs running on that interface; and (ii) the implementation of the recovery capability itself, particularly when the underlying interface provides
both recoverable and unrecoverable objects (i.e. objects for which backward error recovery is and is not available, respectively).

The required primitive features must allow a program to:

(a) establish a recovery point, i.e. a point in time to which the system state can be restored;

(b) discard a recovery point, involving some measure of commitment since recovery to that point will no longer be available; and

(c) recover, resulting in the restoration of the prior state that existed at the recovery point to which recovery is taking place.

Thus, as well as supporting the 'normal' operations on an object, an object manager must also provide support for these recovery primitives. Two schemes for the implementation of backward error recovery have been investigated: the disjoint scheme and the inclusive scheme. The detailed differences between these schemes are described in [2].

More recent investigations of multi-level systems were described in a paper by Tom Anderson, Pete Lee and Santosh Shrivastava[3] which examined the relationships between levels when faults and failures occur. This is of particular importance for reliable systems, since a structured approach to fault tolerance is obviously desirable. For example, an 'abstract' object will be implemented from various 'concrete' objects - faults arising from manipulations of the 'concrete' objects will clearly have little relevance to the user of that abstract object, who should not have any knowledge of the internal implementation of that object. Indeed, it would be most inappropriate to have to cope with all of the faults from 'low' levels at every level in the system.

Our early work on multi-level systems was concerned principally with centralised systems. More recently, we have been concerned with the application of these ideas to distributed systems. This research has taken two forms: the organisation of recovery in distributed systems, and an application of the multi-level model to the protocol organisation in computer networks.

Our first line of investigation, derived from the multi-level system model and carried out mainly by Pete Lee and Santosh Shrivastava, has been concerned with the provision of recovery in decentralised systems. In a distributed system the abstract objects available on an interface may have concrete representations on a number of separate systems (or nodes). For the reasons mentioned previously, we are again interested in the provision of backward error recovery for these objects, and have been investigating the structuring of a distributed system so that this can be achieved. Essentially, the model of computation adopted for this investigation is organised in the manner described for the multi-level model. Every abstract object has associated with it an object manager, that is a program which maintains a concrete representation for that type of abstract object and implements the operations that are permitted on such objects.
While a program can directly access the object managers which reside on the same (local) node, object managers on remote nodes have to be accessed indirectly by sending a message to a process executing on the required node. A 'send message' primitive (and the corresponding 'receive message' primitive) are assumed to be provided by an object manager on the local node to provide the means for accessing non-local objects. A similar problem exists for recovery. While recovery of local objects can be coordinated by the underlying interpreter, it is clear that the interpreter cannot directly invoke the recovery of objects on other nodes. It is therefore necessary to invoke this recovery by programmed actions in the local node. Following from the model of recoverability in multi-level systems, this is simple to organise and achieve, and provides a coherent method for achieving recovery of objects distributed over several nodes. By these means, the abstraction of recoverability on an interface can be provided despite the lack of locality of the objects being manipulated. The details of this implementation of recovery and the ramifications of the disjoint and inclusive schemes are discussed in [4].

It is clear that the consistency of objects in a distributed system can only be maintained while all of the nodes continue their cooperation. If a node crashes or the communications line breaks down, there is a grave danger of inconsistencies being generated in the system, particularly since the recovery capability of a node may be lost. An ingenious protocol called the two phase commit protocol has been developed by several other workers to help overcome these problems and to provide what has been termed "crash resistance". This has also been investigated by Santosh Shrivastava[4] who discusses the extra primitives that have to be provided in a system so that crash resistance can be achieved.

Our most recent work has been concerned with further investigations of the notions of recovery and crash resistance, and has been attempting to find a unified view of these activities since both are concerned with recovery and consistency. Even though this work is still at an early stage, it seems that a unified view is indeed possible, again deriving from our earlier investigation of recoverability. This considered a program establishing and discarding recovery points. The provision of crash resistance also involves the provision of recovery points with the difference being reflected by the manner in which information for recovery purposes is actually recorded and processed. For instance, "ordinary" recovery might involve recording information in volatile storage which would be lost if a crash occurred; crash resistant recovery might involve recording the necessary information on non-volatile storage, and would provide a measure of independence from the ordinary recovery points. While the crash resistant recovery is perhaps the most desirable in theory, in practice cost limitations usually prevent its adoption as the only type of recovery supported. The ramifications of this still requires further investigation.

These ideas form the basis for the current project's first major implementation experiment. This experiment involves providing the abstraction of a recoverable (and distributed) file system to a UNIX process, replacing the standard UNIX file system interface. The
distributed nature of the file system remains visible to each process, which will have to specify the node on which any non-local file to be accessed is held. A detailed design which concentrates mainly on error recovery issues, though it does contain some aspects related to crash resistance, has been completed by Maurice Jegado[5] and is now being implemented. It involves providing each computer in the network with a permanently executing "local file manager" process, which plays the required role of object manager for the files on that computer. A user process, perhaps on some other computer, will access such files via a Distributed Recoverable File Manager which will handle the required inter-processor message traffic. (Further details of the form of recoverability to be provided by this experimental prototype are given in the next section.)

Considerable investigation has been carried out in preparation for this experiment, concerning inter-process message passing in UNIX. The facilities required come in two parts - those for passing messages between processes that happen to exist within the same UNIX system, and those additional ones that are needed when UNIX to UNIX communication is involved. The first of these were implemented last year and have now been transferred to Version 7 of UNIX, on which we are standardising. As regards the facilities for UNIX to UNIX communication, we have experimented with Version 7's uucp program, using an asynchronous line between the two 11/45s. We have not found this satisfactory for our intended usage and believe that Cambridge ring hardware and software will be much more appropriate. Initial development of the distributed file system will be possible using our inter-process message passing systems on a single computer system, but clearly this can only be a temporary expedient.

The second, and as yet much less developed, application of our work on multi-level systems to distributed systems concerns computer networks, and is being undertaken by Fabio Panzieri, under the sponsorship of RSRE Malvern. (The work is reported on in [6].) Briefly, communication protocols provided in systems are usually organised into a hierarchy. For example, at the lowest level, a level 1 protocol might be concerned with the control of a physical communications link and with providing fault tolerance to cope with data corrupted by the transmission - e.g. through the use of CRC checks and retransmission. Thus the lowest level will support a new interface providing, say, packets of data to the level 2 protocol. The level 2 protocol should not be concerned further with the accuracy of the data in the packet - errors in this should have been handled by the level 1 protocol. Level 2 might be concerned with assembling complete messages from sequences of packets, perhaps coping with out-of-sequence packets and faults arising from level 2 buffering limitations, message addressing errors, design faults in the level 2 software, and the like. The interface provided to level 3 might then be in terms of complete messages which have been validated as far as the accuracy of transmission is concerned, and level 3 would be concerned with the 'semantics' of the received data.

This view of the structure of the communications system follows directly from an application of the multi-level model discussed previously, and provides a good exemplification of the model. However, it
seems that in practical systems it is not always the case that such ideal 'reliable' interfaces are provided. Protocols sometimes cooperate to ensure the correctness of the interface behaviour. This is a subject which requires further investigation.

4. System Activity Structuring

Multiple levels, each composed of one or more object managers, constitute a structuring of the actual system. They imply a corresponding structuring of the system's activity. At a given level this system activity will be seen as consisting of the (apparently basic, i.e. effectively instantaneous) invocations of the operations provided by the object managers at the corresponding level of the actual system. However it is highly desirable to identify further structure in the system activity as seen at that level, particularly for system levels which give rise to parallel activity. Such a structuring is provided by the notion of an atomic activity, i.e. a set of operation invocations constituting some portion of a larger activity which, from the viewpoint of the rest of the activity, happens instantaneously. Equivalently, the atomic activity and the surrounding activity are interference-free[7].

As we first described in [8] such atomic activities provide a basis for both forward and backward error recovery strategies in distributed systems. Atomic activities can be pre-planned (e.g. by specifying that a certain program construct constitutes an atomic action) or can arise dynamically. The experimental distributed file system discussed above is to provide backward error recovery based on the use of (a very simple form of) pre-planned atomic activity. These activities are sequential and arise from each of the possibly nested recovery regions that each user process can specify. Atomicity is guaranteed by the object managers using resource locking techniques to control concurrent access to shared objects. A possible extension to the experimental system would allow processes to cooperate in setting up joint recovery regions, so indicating what general pre-planned atomic activities are required - implementation would involve the use of special inter-process message traffic to synchronise entry to and exit from recovery regions, in addition to resource locking.

The problem with dynamically arising atomic activities is finding them when they are needed. This is the problem that our basic work on the chase protocols addressed[9]. This work has been extended recently by Graham Wood. The problem that he has investigated is that of the decentralised provision of recovery control in a distributed system of "optimistic" processes, in which communicating processes establish and commit to recovery points asynchronously with respect to the other processes, and never hold up their execution waiting for the validation of data from another process. The processes are optimistic in the sense that ease of recovery is sacrificed for performance in the absence of error. Recovery control requires the monitoring of information flow and the retroactive fitting of atomic activities on the computation. It performs two functions: to provide recovery of the system to a consistent state in the event of one of the constituent processes initiating recovery action; and to determine when a given recovery point can safely be discarded, i.e. when there is no longer any possibility of
that recovery point being involved in any recovery action.

In this model of computation, the notion of commitment is thus disassociated from the notion of discarding a recovery point. A process makes a commitment not to initiate recovery to a recovery point. It is not safe to discard that recovery point until all potential recovery initiators for that recovery point are committed. A process is a potential recovery initiator for a recovery point if recovery action initiated by that process could induce recovery to the recovery point in question (which may in fact belong to the same process, since the "domino effect" is not precluded).

A relatively simple protocol has been devised to ensure control of recovery in the case where the processes establish a single recovery point and commitment has to be made before another recovery point may be established. Extending the protocol to deal with the case of nested recovery regions has proved to be a difficult task: it seems that the resultant protocol requires the number of messages sent to maintain recovery control to be exponentially related to the number of recovery points in the system.

The other study of atomic activities which has been carried out is that by Elke Best. This has led to the development of a formal model of atomicity based on causal nets, work which was described in one of the series of lectures which he presented at the EEC/CREST Advanced Course on General Net Theory[10]. As outlined below, this work also relates to one problem encountered with techniques such as the chase protocols, which ideally require the recording of just actual information flow rather than the recording of all accesses to shared objects. Such accesses do not necessarily lead to information flow, and so recovery based on access records may be more extensive than is really necessary.

5. Programming Language Issues

The programming language construct corresponding to the operational concept of an atomic activity is termed an atomic action. [11] In our understanding, an atomic action is "that which can be fully described by an effect function". Thus it is always possible to replace an atomic action by another atomic action which has the same effect, without affecting the context in which the action is embedded. In this, an atomic action is a generalisation of a contiguous sequential subprogram of a sequential program. For such subprograms, the property of being replaceable by an effect-equivalent piece of program is provable; if this property is desired in a concurrent environment, however, then it has to be specified explicitly.

The applications of this basic replaceability property include databases, where one of the problems is that the system should always be in a consistent state whenever no transaction is running. This can be achieved by declaring that all transactions are to be atomic and ensuring that transactions by themselves are units of consistency[12]. As another example, consider the proof technique of using invariant assertions. In a concurrent program consisting of atomic actions, it suffices to prove the truth of an assertion initially and to prove its
invariance over all atomic actions separately, in order to deduce its truth upon termination [13].

One beneficial consequence of the replaceability property of atomic actions is that the programmer is encouraged not only to develop well-structured programs, but also to design proofs of correctness in a structured way which corresponds to the structure of the program. Using atomic actions, the proof can in general be divided up in the following way: independently of its environment, an atomic action can be proved to have a certain overall effect, and this can then be used in the proof of a larger part of the program. As part of an effort in gaining experience with the practical use of atomic actions, Elke Best has developed a proof along these lines in [14]. In [15] he has attempted to tie a strictly formal derivation of the proof presented in [13] to the way in which the program is broken up into atomic actions.

Any programming construct needs a formal semantics if it is to be used in formal proofs of programs. The same is true for atomic actions; their semantics in terms of interference-freeness is useful for their understanding and implementation, but not for static verification proofs. Although many researchers have recognised the usefulness of atomic actions, to our knowledge a satisfactory formal semantics has not yet been developed. Because a basic property of atomic actions is the replaceability property, we are interested in semantic formalisms that conceive of operations as input/output relations (rather than, say, sequences of smaller operations). The weakest precondition (wp) formalism of [16] fulfils this requirement. Thus one of our activities has been an attempt to extend the wp formalism so as to include a formal semantics for atomic actions and parallel computation. By progressively weakening the wp framework, some results have been obtained [17] which have led to the characterisation of a restricted form of parallelism and to a semantic clarification of different types of non-determinacy; not so far however to a clear formal semantics of atomicity. (Another outcome of this work was a formal definition of recovery blocks in terms of weakest preconditions, described and discussed in [17].)

This work has strongly indicated that in order to yield a formal definition of atomicity, the wp framework must be extended considerably further than has been done in [17]. In particular, in order to investigate action interdependencies, a state space has to be considered as a structure rather than just a set (as in the wp formalism). Work is in progress to develop this aspect of the semantic framework.

It appears that by continuing work in this direction, a duality can eventually be established between atomic actions and variables: the latter to be regarded as units of information and the former to be regarded as units of information transformation. Initial work in this direction is described in [15]. This work strives for a semantic understanding of information flow in programs, of which there is as yet no agreed definition [18]. So far, only approximations to such an understanding exist [19, 20].

If the analogy between the concepts of variables and atomic actions can be established then this would make atomic actions as fundamentally
significant to programming as variables. (The fact that concurrent pro-
gramming has not until comparatively recently been a subject of inten-
sive study may be the reason why this significance has not been recog-
nised earlier). Both practical and theoretical work is needed to inves-
tigate the precise relationship between these two concepts.

Yet another notion of atomicity has been proposed by some
authors[21,22] who essentially define an operation to be atomic if it
either produces the specified effect or leaves the state of the system
unchanged. This notion of atomicity and that described above seem to
have been equated (or at least not clearly distinguished) by oth-
ers[11,23]. An investigation is being undertaken of the differences,
similarities and significance of the two concepts[24].

Flaviu Cristian in his work on exception handling (begun at Greno-
oble, and now being continued at Newcastle) has also adopted this latter
view of atomicity. In [22] he describes a method for the specification
of abstract data types with operations which are atomic (in the second
sense described above) and with exceptions. Such data types can be used
as basic structuring tools for the design of robust abstract machines
having a well-defined behaviour in all possible circumstances. Issues
concerning the implementation of such data types and a method for vali-
dating the correct implementation of any exceptional effects specified
for operations are discussed in detail in [22]. Earlier project work on
the relationship between exception handling and recovery blocks[25] has
also been continued by providing a unified model incorporating both con-
cepts in a single language supporting data abstraction[26]. In this
model, in contrast to many earlier exception handling proposals, the
exception handling structures are compatible with other language defined
control structures. The aim now is to extend this model to deal with
asynchronous systems, to which end a study of object-oriented languages
such as Smalltalk has been started. In addition, an investigation of
the possibility of systematic identification of exceptions is being purs-
sued, with the hope of providing guidelines for the rigorous derivation
of acceptance tests by static analysis methods.

In contrast to the considerable progress made on the rather funda-
mental programming language issues described above, our early attempts
at defining a useful notation for specifying system-wide reliability
measures and constraints proved to be somewhat premature. A survey of
various proposed languages for so-called 'programming-in-the-large' and
for specifying module interconnections was carried out, but it now seems
clear that current programming language research by various groups on
object-oriented languages is in due course more likely to provide an
appropriate basis for our work on this topic.

6. Other Activities

Four other activities are worthy of mention. Construction of the
PDP-11 recovery cache has been completed by Din Ghani, Keith Heron and
Pete Lee, who described this work in a paper given at FTCS-9[27]. The
device sits astride a Unibus, and can be set up by a processor (which
sees it as a peripheral) so as to intercept operations writing to a
chosen area of memory. By this means the contents of any words that are
about to be overwritten for the first time since a recovery point was established can be automatically saved for recovery purposes. Initial performance measurements indicate that the recovery cache will slow down all bus transfers by approximately 10%, with write transfers incurring an extra 600 nsec overhead. Some minor hardware changes are now in progress and the cache awaits its first major test, which will probably take the form of a student project involving the implementation of a sophisticated device driver.

The Recoverable Concurrent Pascal system [28] has provided a basis for some interesting experiments in fault-tolerant system design. The most successful was by Eric Saltzman, whose M.Sc. dissertation was on the design of a model transaction processing system, one of the requirements for which was that it should never cause its data base to be lost or corrupted despite design faults in the programs (it satisfied this requirement!). The Recoverable Concurrent Pascal system has been distributed to TRW in Los Angeles and to Prof. K.H. Kim at SUNY Binghampton.

Some work on fault tolerance in real-time systems was initiated by Tom Anderson whilst he was on study leave at the NASA Langley Research Center, and was concerned with the high level of reliability that is often required when real-time computer systems are used to control critical applications. It might be expected that attempts to increase the reliability of such a system by employing techniques for tolerating faults of design would be severely hampered by the difficulties created by concurrent activities and timing constraints. However, an investigation conducted (in conjunction with John Knight of NASA Langley) has concluded that a relatively simple and pragmatic approach can provide effective fault tolerance in many real-time systems. The proposed approach assumes that the synchronisation and timing constraints of a real-time system can (in most cases) be adequately described by a "synchronisation graph", which models the process structure of the system as an acyclic graph. In order to simplify the provision of backward error recovery severe restrictions are imposed on interprocess communication - concurrent processes may only communicate if they have the same completion deadline. A simple classification scheme has also been suggested for assessing the spread of damage between processes, with a corresponding hierarchy of responses intended to enable the system to continue operating[29].

The fourth activity consisted of an experiment by Elke Best and Fabio Panzieri on the problems of designing distributed systems. This resulted in the devising of a fail-safe algorithm which enables N nodes connected in a ring to synchronise with each other even if one out of a class of anticipated failures occurs[30]. It is hoped that this algorithm can be extended and implemented efficiently on the Cambridge ring hardware which the project is expecting to receive.

7. Exploitation and Interactions

The two week Advanced Course on Computing System Reliability that we organised at Newcastle in the Summer of 1978, under the sponsorship of the SRC and the EEC/CREST, attracted the full complement of
approximately eighty attendees, a considerable proportion of whom were from industry. The repetition of the course in 1979 at the Santa Cruz campus of the University of California had a smaller attendance, but with a larger proportion from industry. The lecture notes from these courses have now been published by Cambridge University Press[31]. Specially commissioned versions of this course have been requested by several industrial organisations. The first of these, a one week residential course for ICL system designers, will be held in Newcastle in September 1980. This is in fact part of a planned formal collaboration with ICL on the formulation of policies concerning system reliability goals and strategies, initiated by Mr. D. Pickard, Manager of the Engineering and Diagnostic Systems Division of ICL. To date this collaboration has taken the form of a document interchange agreement and a joint two-day workshop sponsored by ICL at which more specific collaborations between the project and ICL were discussed.

Our planned collaboration with CAP has so far taken the form of a one-day workshop and work on a detailed proposal to a U.S. government agency for a project to be carried out at the Microelectronics Applications Research Institute (MARI), the Newcastle-based contract research institute set up by the University, the Polytechnic and CAP. This proposal has, we understand, been well received, though budget approval is not yet forthcoming. The plan is for some suitably qualified and experienced CAP personnel to try out our fault tolerance techniques on a model of an application which is of interest to the agency, under the guidance of reliability project members. However other forms of collaboration are also envisaged, such as reliability project involvement with a suitable existing CAP project, providing that client approval can be obtained.

RSRE Malvern's interest in our project has of course taken the very constructive form of sponsorship of an additional research associateship, which is held by Fabio Panzieri. He has the dual role of taking part in the main work of the project, and of investigating the relationship between fault tolerance in distributed systems and in the sorts of computer networks that are of relevance to RSRE. Incidentally, it is at RSRE that the first implementation of a distributed error recovery system based on our chase protocol scheme, has been made.

Interest in our work on fault tolerance and recovery has recently come from the Research Department of the Post Office Telecommunications Division who are (presumably) working on similar ideas for incorporation into their computer controlled telephone switching systems. While interactions between this group and ourselves is at an early stage, it is hoped to form some more formal and fruitful arrangements for collaboration. It is perhaps of interest to note the recently revealed decision by the Post Office Telecommunications Division to establish one of their two pilot Software Engineering groups at Newcastle, the choice of site being at least in part due to their wish to establish links with the Computing Laboratory in general and the Systems Research Group in particular.

There are many other groups, particularly in the USA, working on topics related to ours, and with which the reliability project has
maintained close contact. Included in this group are SRI International, Xerox PARC, NASA Langley, TRW, MIT Laboratory for Computer Science, University of Illinois at Urbana-Champaign, DEC, LAAS Laboratories at Toulouse and IBM San Jose. Several project members have visited and presented talks to these (and other) groups, and many have reciprocated by visiting us at Newcastle. In fact, over the last two years we have had visits from 15 people (from 6 countries) working on related topics, while project members have given (at least) 50 seminars on various aspects of our work, as well as the lectures forming major parts of the two Advanced Courses on Computing Systems Reliability and General Net Theory.

8. **Project Hardware**

The hardware inherited from the previous reliability project consisted of two PDP11/45 systems, only one of which was capable of running the Unix operating system. Recent enhancements to this equipment to provide extra disk and main memory now enable version 7 of Unix to be run on both systems. In fact, we now run Unix permanently on one system, with the other being available for other systems such as the Recoverable Concurrent Pascal system and for work on the hardware recovery cache. The Unibus switching equipment which was developed by Keith Heron for the earlier project has proved to be of significant value for the dynamic reconfiguration of our equipment which is still required.

Provision was also made in our hardware budget for the purchase of three work stations which are, with the two 11/45 systems, to form a small distributed system for our experimental work. The arrival of the first of these stations, based on the LSI 11/23, is now imminent. Delays in its purchase were caused mainly by uncertainties about the 11/23 and Version 7 of Unix which we want to run on it, and by the long delivery dates of the hardware. Purchase of the second two work stations, also to be based on 11/23s, is being delayed until experience with the first station has been gained. We also expect this delay to be beneficial in that recent changes in disk technology, with the introduction of Winchester type disks, will provide us with a much better system than the floppy disk based first work station.

The availability of equipment from the DCS pool has also proved to be of significant use to the project. The two VDUs, the two Teraks, the diablo printer and the magnetic tape unit are all being heavily used, particularly the tape unit which has much simplified the distribution of software.

As discussed above, we have found that one possibility for interconnecting our Unix systems (that based on asynchronous lines and the Unix uucp program) has not proved to be satisfactory and we have requested Cambridge Ring hardware from the DCS pool.
References


9. P.M. Merlin and B. Randell, "Consistent State Restoration in Distributed Systems," Digest of Papers FTCS-8, pp.129-134 (June 1978). (Also TR113, Computing Laboratory, University of Newcastle upon Tyne)


17. E. Best, "Notes on Predicate Transformers and Concurrent Programs," TR145, Computing Laboratory, University of Newcastle upon Tyne (1980).


27. P.A. Lee, N. Ghani, and K. Heron, "A Recovery Cache for the PDP-11," Digest of Papers FTCS-9, pp.3-8 (June 1979). (Also TR134, Computing Laboratory, University of Newcastle upon Tyne)


### APPENDIX I

<table>
<thead>
<tr>
<th>TR127</th>
<th>SHRIVASTAVA, S.K.</th>
<th>1978</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Concurrency Pascal with backward error recovery.</td>
<td></td>
</tr>
<tr>
<td>TR130</td>
<td>BEST, E.; RANDELL, B.</td>
<td>1978</td>
</tr>
<tr>
<td></td>
<td>A formal model of atomicity in asynchronous systems.</td>
<td></td>
</tr>
<tr>
<td>TR134</td>
<td>LEE, P.A.; GHANI, N.; HERON, K.</td>
<td>1979</td>
</tr>
<tr>
<td></td>
<td>A recovery cache for the PDP-11.</td>
<td></td>
</tr>
<tr>
<td>TR137</td>
<td>ANDERSON, T.; LEE, P.A.</td>
<td>1979</td>
</tr>
<tr>
<td></td>
<td>The provision of recoverable interfaces.</td>
<td></td>
</tr>
<tr>
<td>TR145</td>
<td>BEST, E.</td>
<td>1980</td>
</tr>
<tr>
<td></td>
<td>Notes on predicate transformers and concurrent programs.</td>
<td></td>
</tr>
<tr>
<td>TR147</td>
<td>BEST, E.</td>
<td>1980</td>
</tr>
<tr>
<td></td>
<td>An operational characterisation of the atomicity of activities.</td>
<td></td>
</tr>
<tr>
<td>TR149</td>
<td>SHRIVASTAVA, S.K.</td>
<td>1980</td>
</tr>
<tr>
<td></td>
<td>Structuring distributed systems for recoverability and crash resistance.</td>
<td></td>
</tr>
<tr>
<td>SRM No.</td>
<td>Title</td>
<td>Authors</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
</tbody>
</table>
217. A Data Flow Computer with Addressable Memory. 
R.P. Hopkins, 
R.W. Rautenbach, 
P.C. Treleaven, 
6 April 1979.

218. Software Fault Tolerance. 
B. Randell, 
12 April 1979.

219. Some Unix Programs. 
P.A. Lee, 
22 May 1979.

B. Randell, 
11 June 1979.

221. Weakest Preconditions and Recovery Block Semantics. 
E. Best, 
18 June 1979.

F. Panzieri, 
6 August 1979.

P.A. Lee, 
10 August 1979.

B. Randell, 
2 October 1979.

F. Panzieri, 
1 October 1979.

226. A Message Passing System for UNIX System(s). 
P.A. Lee, 
18 October 1979.

S.K. Shrivastava, 
23 October 1979.

M. Jegado, 
P.A. Lee, 
F. Panzieri, 
1 November 1979.

T. Anderson, 
5 November 1979.

230. Notes on Computer Network Reliability (again!) 
F. Panzieri, 
3 December 1979.

L.F. Marshall, 
12 December 1979.
<table>
<thead>
<tr>
<th>No.</th>
<th>Title</th>
<th>Author</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>232</td>
<td>Uniqueness of the Inconsistency Set.</td>
<td>E. Best,</td>
<td>10 January 1980.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F. Cristian,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F. Panzieri,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F. Panzieri,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 April 1980.</td>
<td></td>
</tr>
<tr>
<td>246</td>
<td>Trip Report: Course on &quot;Data Types - Concepts and Applications&quot;, IRIA, France, 17-21 March 1980.</td>
<td>F. Cristian,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8 April 1980.</td>
</tr>
</tbody>
</table>


249. The Reliability and Integrity of Distributed Systems.

250. Document Preparation of V7 Unix.


254. A Distributed Recoverable File Manager for UNIX Systems.

E. Best, M.W. Shields, 28 April, 1980.


B. Randell, 2 June 1980.

F. Panzieri, 5 June 1980.


M. Jegado, 26 June 1980.