A MODEL FOR INVISIBLE RECOVERY APPLIED TO COMMUNICATIONS NETWORKS

GEORGE G. MITCHELL STEPHEN BROWN

Department of Computer Science,
National University of Ireland, Maynooth.
Maynooth,
Co. Kildare.
Ireland.
{georgem, sbrown} @cs.may.ie

ABSTRACT

In this paper we present the advances we have made in our network fault tolerant computing project [1]. We examine the problems of failures within network communications and telecom systems and outline the usefulness of our localised Invisible Recovery (IR) solution. We evaluate the current techniques [2,3,8] that provide recovery within distributed / network systems; by doing so we then derive a table of comparison that we use to pinpoint the different approaches taken to providing recovery. With this table we then construct state transition models in UML that faithfully reproduce the operation of systems when in failure and failure free states. These state models are replicated for all the techniques that we examine and we contrast them with our IR technique.

Our IR approach makes use of an envelope surrounding an unchanged software protocol layer and operates in a Super Server fashion modelled on the Inetd daemon common in UNIX systems. This envelope stores just sufficient information on each connection to enable an invisible recovery of the protocol layer after a software fault. IR provides the means to permit a reconnection to take the place of the original connections. As a result of knowing the necessary attributes that a recoverable network system should have, including the required state information, our technique allows for the near continuous operation of a network system using software fault tolerance. We investigate the problems for the integration of the IR technique into existing network protocols. We also explore the use of recoverable techniques on the grounds that they can be used as a means of providing solutions to failures due to malicious attack or software faults.

We believe that with the aid of state modelling we have refined our IR technique and have developed a system that can provide non-invasive fault tolerance to existing network protocols. Our IR technique has particular applications in the high-volume consumer-technology market where cost is paramount but where reliability is equally important.

KEYWORDS

Recovery, Fault Tolerance, Communications Networks, Reuse.

INTRODUCTION

Through the use of use modern software design methodologies reliability of software has been improved, unfortunately experience [4,5,6,7] has shown that the use of these alone is inadequate. When dealing with modern Safety Critical Systems such as those in Avionics Control Systems and those systems that must integrate with an existing legacy system, software fault tolerance is invaluable.

One important thing to note about the implementation of fault tolerant systems is that they generally may be implemented in two forms, the most predominant form is that of code invasive *design time* fault tolerance, the second and also the far smaller is *non code invasive maintenance* fault tolerance. The fault tolerant technique, which has been used in this project, has been a non-code invasive type and this has enabled us to extensively reuse software. . It is notable that one should strive for the least code invasive mechanism as this clearly reduce the potential for the number of possible faults that could be injected in the system. We will examine our technique from a couple of differing sides, from the current research side and also from a state diagram approach so that we can fully understand the workings of the system.

With these concepts in mind let us now focus on the issue of network systems, the provision of fault tolerance to network systems is one which primarily explores the problems of hardware failures and also crashes system. The problems of poor connections resulting in loss of data and or continuous reconnection of the underlying protocol daemons has to date been poorly explored. Development of fault tolerant network systems without a consideration for the similarities between the current protocols results in a diversity of fault tolerant mechanisms being employed. We have developed a technique for network communications recovery through the use of a component, *IR* Invisible Recovery, which is applicable across a wide variety of network protocols. Initially we examined UDP over IP, but this we believe is only the start of the design of a system that will have a wide scope of use. Another area in which the use of fault tolerant systems is in the network defence and security field, by providing a staggered reduction in the operability of the network rather than a complete failure users manage the shut down of the system in an orderly manor in the event of a fail which is impossible to recover from.

RELATED RESEARCH

The design and implementation of software fault tolerance has taken a number of different approaches. Each of which has provided improvement and extensions to the ever-expanding requirement for reliable software. The use of acceptance tests at points that the reasonableness of a result is in question, through to the use or redundant systems have all helped in the provision of this fault tolerance. What we are particularly interested in is those techniques that make the most use of the existing system that we are attempting to improve the reliability of through the use of fault tolerance. Therefore software reuse techniques coupled with fault tolerance attract our attention, it is interesting to note that communications RFC's in the area of fault tolerance are directed at developing new protocols that provide reliability to the protocol through code invasive techniques rather that stand alone non code invasive techniques as we discussed in Mitchell 2000.

Probably the most significant related research in the area of fault tolerance applied to processes and also protocols is in message logging and checkpointing techniques such as those developed and used by D.B. Johnson [8] and Elnozahy [3]. Using the well-developed checkpointing technique together with a logging system which both allows for forward recovery and backward recovery enables the systems to provide a high degree of reliability to the communication system. Johnson designed two main techniques *pessimistic* and also *optimistic* message logging.

For pessimistic message logging a new sender-based message logging protocol was developed. Each message was logged in a local volatile memory of the sending machine and ordering of the received messages was organised by a receive sequence number. Logging of messages overlapped the execution of the receiver until the receiver attempted a new send message. With this form of fault tolerance applied to a communicating protocol system applications had an overhead under 16 percent and an average overhead measured 2 percent or less depending on the size and communication intensity.

Optimistic message logging outperformed pessimistic logging since the logging occurred asynchronously. Johnson presented a new optimistic message logging system that guaranteed to find the maximum possible recoverable system state which had not to-date been previously attained by other optimistic methods. All messages *and* checkpoints are utilised in his method and thus some messages received by a process before checkpointing were not necessarily required to be logged. With this technique overhead was kept to between 1 and 4 percent.

INVISIBLE RECOVERY

Our invisible recovery technique has three separate and distinct parts:

1. Timing
2. Restart
3. Protocol Mangling

1 Time:
Crucial to the correct operation of many network systems is the timing both of packets and the overall session. The OSI protocol stack introduced session layers that provided a compensation operation for intermittent connections by token management and check-pointing data transfer. This permits multiple network sessions to be integrated together to form one continues data transfer. This type of method is not available in all network and telecom protocols for example, TCP / IP, the timing and also the non-interrupted communication between networked machines is necessary for the continues operation of the protocols. By providing a proxy server *IR* to the client we maintain the acknowledgement of data packets and also enable the system to disguise the fact of restarting other servers, all of this is accomplished in under a couple of seconds when the entire system is in a non congested state, to understand the actual operation of *IR* it is necessary to examine its two fundamental parts; Restart & Protocol Mangling.

2 Restart:
The initial communication required for a connection between networked computers normally requires the

issuing a number of protocol data units (PDU). Each of these are used to enable the client to register itself with the server, typically 3 to 5 PDU' s are required, in the TCP protocol, 3 PDU' s are used in the well know 3-way hand shake. Our IR system operates in the form of an envelope that surrounds an unchanged protocol entity, with this design it is possible to operate between upper and lower network layers providing a "restartable" component for most telecommunications and network protocols. Our envelope makes a data copy of all data that passes through the layer. On detection of a fault our restart component reinitialises the connection by reissuing PDU' s and then creating a new connection. Our envelope, running as a process, uses a database containing the state information of the application process and also the protocol process. Typically this consists of; port number, IP address, unacked_data, last_seq_number, last_acked_request, etc. This provides sufficient information to enable the IR process to restart and to then hide the failure of the underlying network entity, we accomplish this with the aid of *protocol mangling.*

3 Protocol mangling:
This is essentially a mapping of all the last known good state information to the current new state information. It is necessary so as to cope with the newly regenerated protocol entity that will have been "reincarnated" with differing sequence numbering, possibly different port information and possibly incorrectly matched data. We use the stored data to correct the data problems of acknowledged verses unacknowledged data. Then by mangling the state information such as old_port address to new_port addresses, old_ip to new_ip and so forth we make it is possible to disguise the fact of a protocol entity failure, and continue to operate correctly.

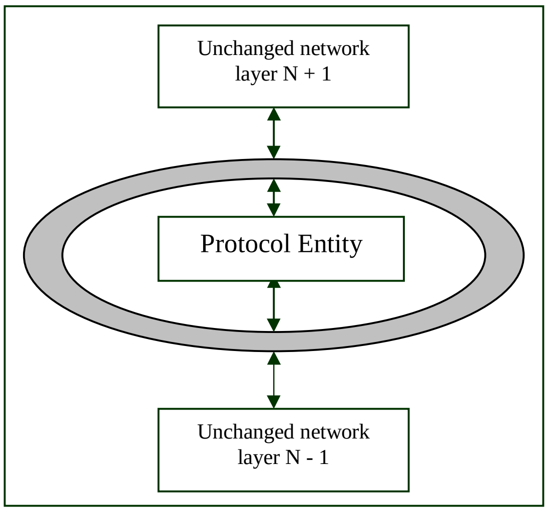


Figure 1. IR envelope communication

STATE DIAGRAM FRAMEWORK

Taking a very simplified initial example we can view a communication between two communicating processes **p** and **q** (figure 2), we introduce a token datagram which we pass backwards and forward between the two communicating processes using the sockets **c** and **c`**.

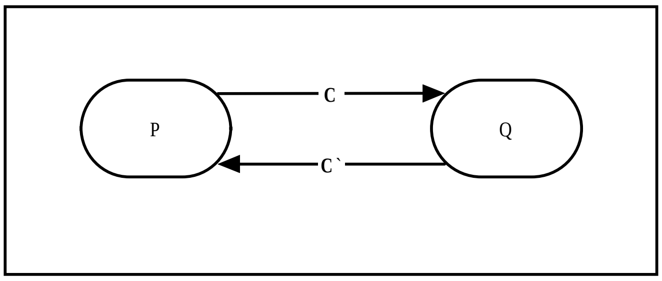


Figure 2. Simple Network example sockets c & c`

We can describe the internal operation of the p process as two states s0 and s1 and a receive and send operation enabling transition (figure 3)

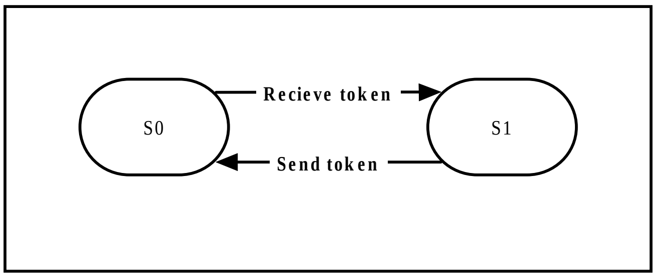


Figure 3. State of process p.

With the simple notation of the network we can introduce the Invisible Recovery (IR) process which we have designed to provide fault tolerance in communicating systems. IR is positioned logically between the communicating processes as in figure 4.1. With this understanding of the logical layout of the IR process we can now develop our model to fully encompass the state transitions of the network operating with IR. We now present the operation of the network system with IR in a failure free environment. Firstly we specify that the network consists of processes P,Q,IR, four socket connections and a single systems call for the determination of the Process State of Q.

In state diagram 4.1 we see the initial state of the system, with p in state S1 i.e. with control. Stated diagram 4.2, control has left p and therefore p now has state S0. A token (message, packet or datagram) has been issued to IR and is in transit within the socket, in state diagram 4.3 the token is received by IR and changes the state of IR from S0 to S1. When control passes to IR a series of operations occur which we will examine in more detail later in this paper, initially let use just allow IR to act as a message forwarding server. State diagram 4.4 show the sending of token to process Q, control passes from IR.

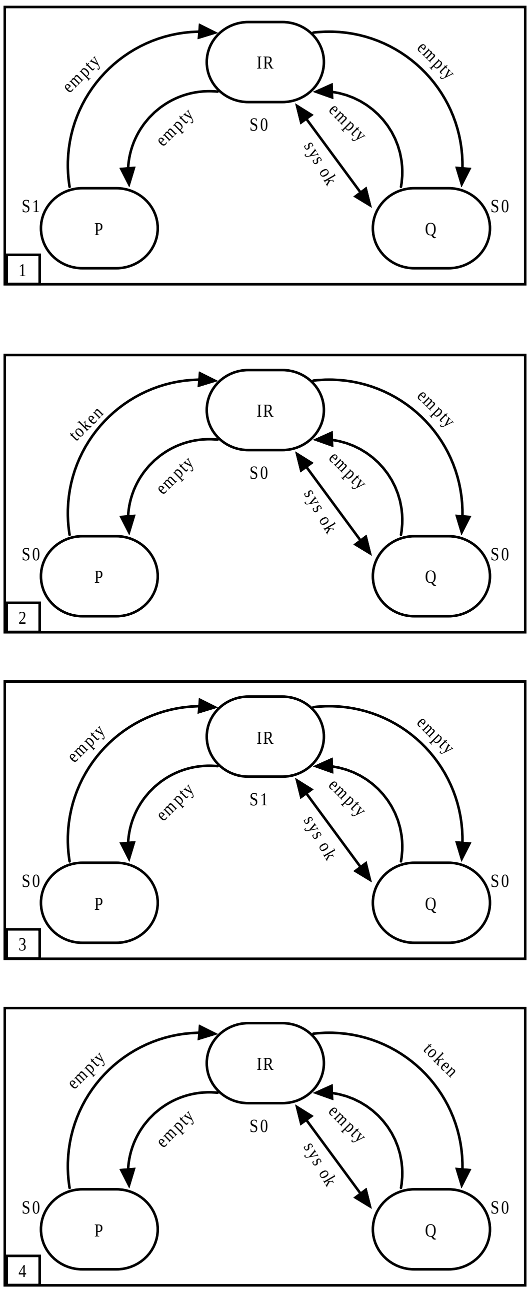


Figure 4. State diagrams 1 to 4 of network with IR

In state diagrams 5 through 8 we conduct the same operations looking at the return of the token following its receipt by q. This simplified model demonstrates clearly the operation of IR within a communicating system. To understand the inner operation of IR we now expand the IR process state diagram and in figure 6 we examine the steps that IR takes in the forwarding of messages. State diagrams 6.1 to 6.4 outline how IR checks first for the status of Q the server process and if following a successful outcome to this query IR proceeds to send the message to Q and then receive the resulting response from Q.

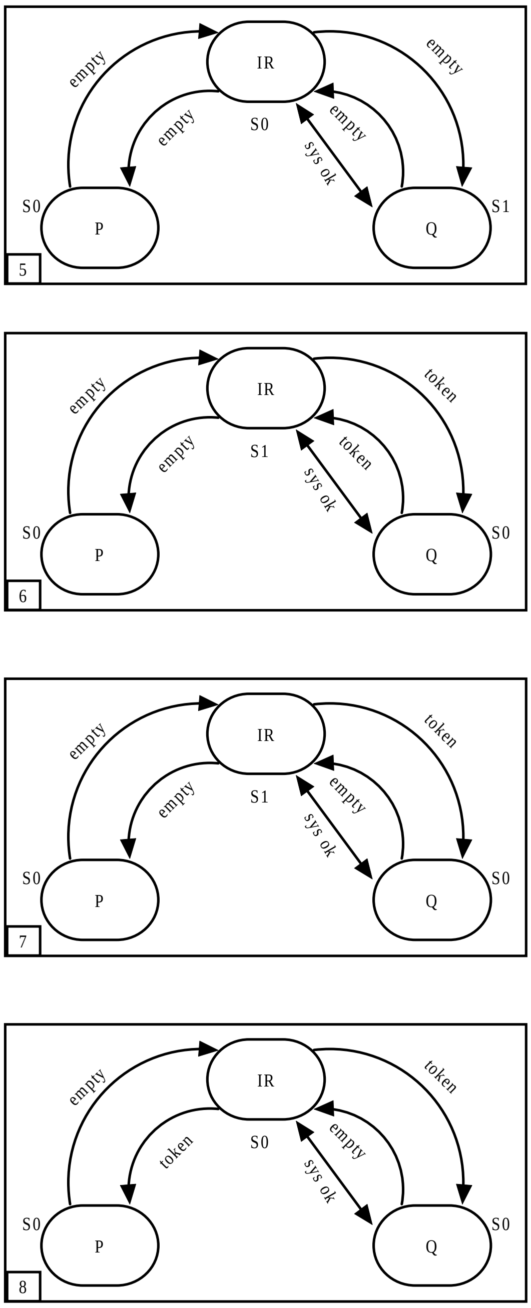
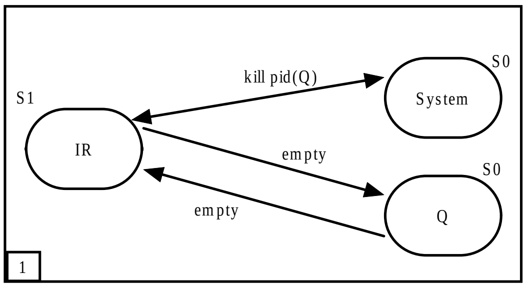
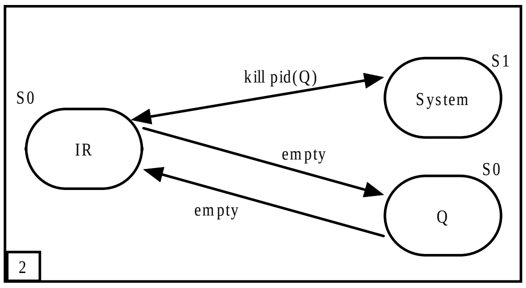


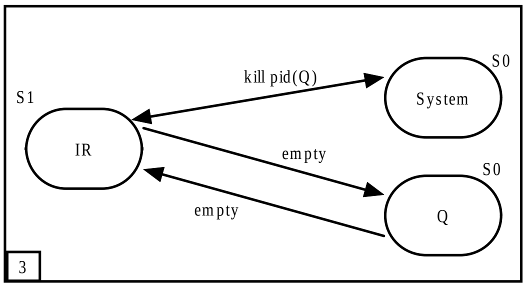
Figure 5. State diagrams 5 to 8 of network with IR

State diagrams 7 outline how IR detects a failed server process, since *IR* is styled on the inetd super server it is easy for the IR process to detect the failure of a child server process and as such take the appropriate action in storing incoming messages form P and also restarting the failed process. Let us now examine how *IR* contends with this. Initially we sate the operations that are performed in the IR process as follows;

- ❖ Receive message from p
    - ❖ Check status of server process (Q)
    - ❖ If Q OK send message
    - ❖ Else save message to store
    - ❖ Restart failed server
    - ❖ Re-send all save messages form store.
    - ❖ Return to normal operation.







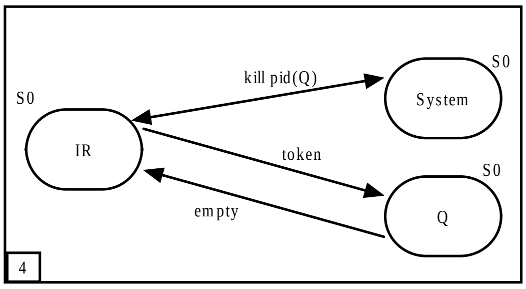
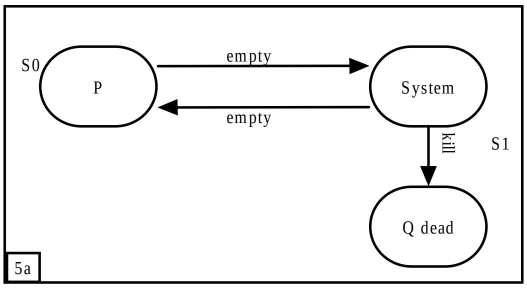
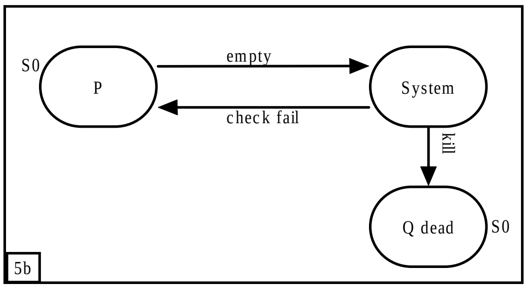


Figure 6. IR process state diagrams 1 to 4.

From these diagrams we can see that there is little if any cost introduced into the communicating process. Although IR acts as a message forwarding system no memory caching is necessary and therefor the transit across the IR process along with the extra two sockets is the entire overhead of the system this too is on a complete round trip. If we now introduce the state transition diagrams that represent the workings of IR while in a state that permits failures, we see the extra states necessary for the detection and reconfiguration of the communicating system.





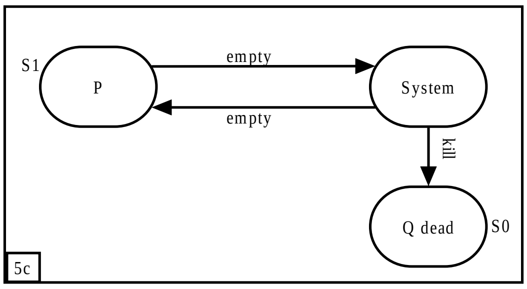


Figure 7. IR detecting crashed server.

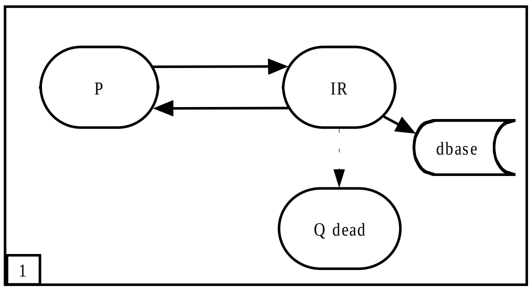


Figure 8. IR operating in fail state with dbase.

Figure 8 above show the logical lay out of the dbase which is used to record incoming messages from P while Q is being restarted and initialised. Figure 9 outline the restart and re-sending of messages stored on the dbase while Q was being reinitialised. Once the store is emptied IR returns to its standard failure free operation and continues to do so until a terminal error e.g. shut down, power failure etc.

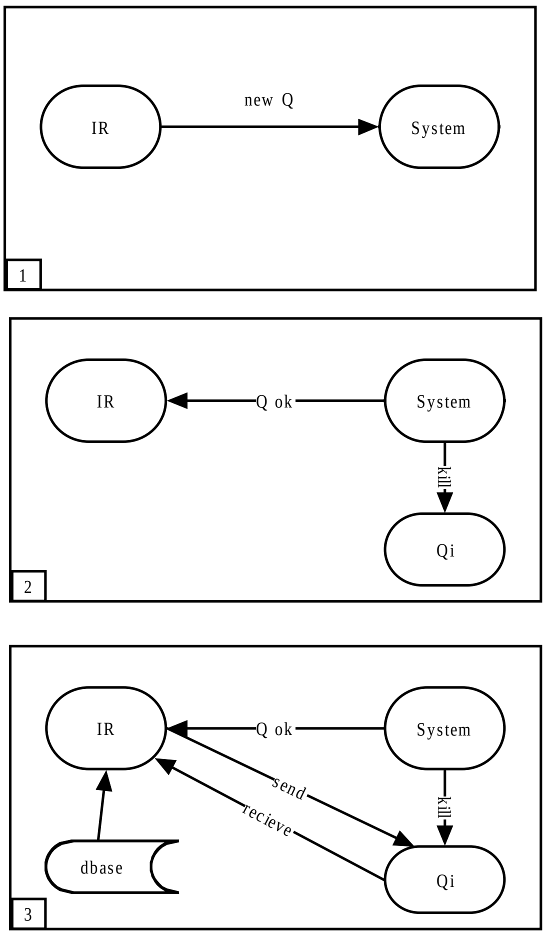


Figure 9. IR restart

CONCLUSION & FUTURE WORK

IR appears to be successful and now we intend to examine its usefulness against other protocols such as DSL. It is our view that IR can be used in a wide variety of roles such as those depicted in figure 10.

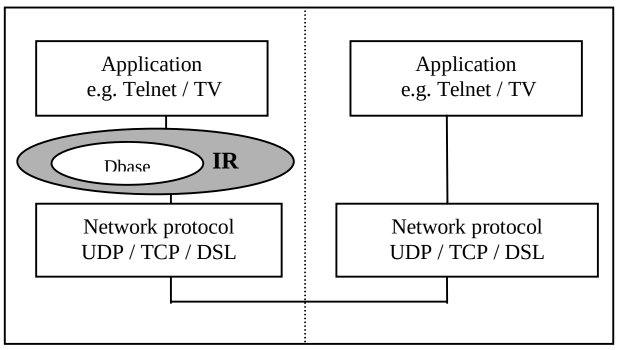


Figure 10. IR system communications with other protocols

We intend to validate each of these applications with testing and formal verification particularly for significantly different protocols such as ADSL or xDSL. With the increased interest in providing digital television to the domestic market our technique should enable the recovery of such systems in a fast and low cost manner.

REFERENCES

[1] Mitchell, G.G., Brown, S., "An Approach for Network Communications Systems Recovery", Applied Informatics 2000, Innsbruck, Austria, pp 575-578, 0-88986-280-X, 2000.

[2] Johnson D.B., Personal communication. June 2000.

[3] Elnozahy, M., Alvisi, A., Wang, Y., Johnson D.B., "A Survey of Rollback-Recovery Protocols in Message-Passing Systems" School of Computer Science Carnegie Mellon University, Pittsburgh, PA 15213 USA. CMU-CS-99-148, June 1999.

[4] A. Spector & D. Gifford, *The Space Shuttle Primary Computer System*, Communications of the ACM, 27(9), 1984.

[5] B.Randell, *System structure for software fault tolerance*, IEEE Transactions on Software Engineering, SE 1, June 1975, 220-232.

[6] P.A. Lee, and T. Anderson, *Fault Tolerant Principles and Practice 2 ed*., (Wien - New York: Springer - Verlag, 1990).

[7] M.R. Lyu, *Software Fault Tolerance*, (New York: John Wiley and Sons Inc, 1995).

[8] Johnson D.B, “Distributed System Fault Tolerance Using Message Logging and Checkpointing”, Rice University, Houston, Texas, December 1989.