International Workshop on Computer-Aided Design, Test, and Evaluation for Dependability,
Beijing, China, July 2-3, 1996, pp. 66-71.

An Integrated Environment for Development and Testing of Software Fault Tolerance Systems

Kam S. Tso and Eltefaat H. Shokri
SoHaR Incorporated
Beverly Hills, CA 90211
{tso,shokri}@sohar.com

Abstract

The need to experiment with software fault tolerance to investigate its effectiveness and the desire to reduce repetitive effort in creating experimental SWFT systems have motivated the development of an integrated testbed environment which eases the construction and evaluation of software fault tolerance systems. The integrated environment comprises a library of reusable components from which a wide variety of SWFT systems can be built. In addition, interface components supporting the seamless integration of applications into the SWFT systems, and fault-injection components supporting dependability evaluation are also provided. Another important ingredient of the environment is a set of graphical tools to facilitate the creation of SWFT systems from the reusable components, monitoring the execution of the resulting SWFT systems, and evaluation their dependability through fault-injection testing.

1: Introduction

Although software fault tolerance (SWFT) was proposed more than two decades ago, it is not widely used partly because its effectiveness is still controversial. Many experiments have demonstrated favorable results when SWFT is properly applied [2, 4]. However, a few theoretical and empirical studies have raised concerns on the validity of the design diversity approach which conjectures that independent development can minimize the probability of common-mode errors [7, 12].

Experimentation in SWFT techniques is hindered by the difficulty of incorporating them in complex systems. Moreover, the majority of SWFT techniques are not transparent to the application developer, meaning that the developer has to address the concerns of the chosen SWFT scheme together with the application. This in turn increases the complexity of the system to be developed.

The need to experiment with software fault tolerance to investigate its effectiveness and the desire to reduce repetitive effort in creating experimental SWFT systems have motivated our effort in developing the Reusable Software Fault Tolerance Testbed (ReSoFT), a testbed environment for the construction and evaluation of software fault tolerance systems.

The rest of the paper is organized as follows: Section 2 gives an overview of the integrated testbed environment. The identified reusable components are described in Section 3. Section 4 presents the graphical tools for SWFT system integration, SWFT system monitoring, and fault-injection testing. Section 5 discusses the implementation of the distributed recovery blocks system within the integrated environment. The last section provides a conclusion of the paper.

2: Integrated Environment for the Reusable Software Fault Tolerance Testbed

The integrated environment of ReSoFT provides (1) a library of reusable components from which software fault tolerance systems can be built, and (2) a set of graphical tools facilitating the construction of such systems, integration of applications, monitoring execution status, and evaluation their dependability. Figure 1 shows an overview of the integrated hardware and software environment.

The hardware platform of the testbed, shown in Figure 2, is a network of Sun workstations running the Solaris operating system. The widespread use of PCs has also prompted an ongoing effort to port the environment to the PC/Windows platform. We have chosen to use an open architecture employing commercial-off-the-shelf hardware and software products because they are widely available and can evolve along with future technological changes.

As fault detection and recovery mechanisms rely greatly on the reliable communication among the computing nodes, the network becomes the single point of failure. In order to

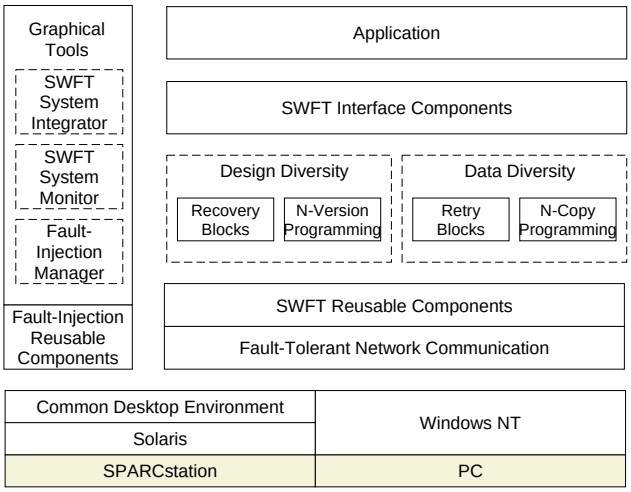

Figure 1. Integrated Environment for Software Fault Tolerance

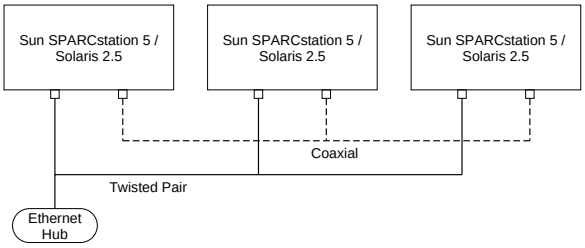

Figure 2. Hardware Configuration of ReSoFT

tolerate network hardware and software faults, the workstations are connected by dual-redundant Ethernet networks. One uses twisted pair while the other uses coaxial cable for physical connection.

As shown in Figure 1, the software environment running on top of the hardware platform consists of a number of layers. The bottom layer is the network communication which provides fault-tolerant communication over the redundant networks. On top of it are the reusable SWFT components from which SWFT systems can be built. Currently two SWFT techniques have been implemented — *design diversity* which makes use of diverse techniques in design and implementation [3], and *data diversity* which makes use of diverse data inputs [1] — for tolerating software faults. Reusable SWFT components have been developed for these two techniques as exemplified by the recovery blocks, N-version programming, retry blocks, and N-copy programming schemes. SWFT Interface components are used to isolate application-specific functions from the reusable SWFT components and to facilitate the integration of applications to the SWFT systems. In additional to the reusable components, the ReSoFT environment also provides a set of graphical tools to ease the construction, monitoring, and testing of SWFT systems.

3: Reusable Components

An in-depth object-oriented analysis of the well-established SWFT schemes has been conducted based on the Booch method [5] for identifying reusable SWFT components. During this process, care was taken to make sure that (1) the components capture the common functionality of a wide variety of SWFT schemes, and (2) application-specific functions are identified and excluded from the generic reusable components. However, application-specific functions are derived from generic classes using the inheritance mechanism. The result of the analysis is a reuse framework of software fault tolerance and the details were reported in [15].

Based on their functionality, identified components can be classified into the following categories: SWFT executive components, SWFT support components, SWFT interface components, Network communication components, and Fault-injection components.

The reusable components are implemented using Ada95, the revised standard of the Ada language [9]. Ada95 increases the flexibility and applicability of Ada by introducing new features supporting object-oriented programming, hierarchical libraries, and development of real-time systems, while retaining its reliability goal. The details on how the new Ada95 features support software reuse and real-time processing in implementing the ReSoFT integrated environment can be found in [14].

3.1: SWFT Executive Components

The SWFT executive components are responsible for managing orderly execution of the SWFT schemes which include the following responsibilities: (1) initialization of the execution including the activation of the various tasks, (2) activation of the next execution cycle, (3) managing orderly execution of the ongoing cycle, and (4) producing an acceptable result whenever possible. Depending on the complexity, there can be one or more executive components for each scheme.

3.2: SWFT Support Components

Comparing existing SWFT schemes, we find several generic fault tolerance related components employed repeatedly. We designate them as SWFT specific support components. Seven types of reusable SWFT Support components have been identified during domain analysis: *Try Block*, *Acceptance Test*, *Voter*, *Heartbeat*, *Watchdog*, *Checkpointing*, and *Data Re-expression* components. The

Heartbeat Component is added for faster fault detection. Heartbeats are periodical messages which are exchanged among the nodes participating in the SWFT scheme and used for identifying the health status of participant nodes.

3.3: SWFT Interface Components

The SWFT components should be designed such that they can be reused by various applications. However, there are aspects which vary from one application to another. The inheritance mechanism provided by object-oriented methods has been used to separate application-specific concepts from application-independent ones. The SWFT components are designed as abstract classes and the application designer creates subclasses for adding the application-specific functions. These subclasses are part of the SWFT Interface component category. Moreover, the SWFT Interface component category also includes components which are used to decouple applications from the SWFT components. The *Application-Specific Acceptance Test* and *Application-Specific Checkpointing* components belong to interface components.

3.4: Fault-Tolerant Network Communication Components

The Network Communication components provide an application transparent reliable inter-node communication among the ReSoFT nodes. The message redundancy management is designed to be transparent to the users. It automatically detects the existence of a second network and establishes an alternate connection over it. Fault-tolerant communication is achieved by sending duplicated messages, each with the same sequence number, over both connections. The *receiver* can thus recognize and discard redundant messages. Since the first message is forwarded to the upper layer, a message delay in the other network will be tolerated. If a network is permanently failed, the failure will be detected by missing consecutive messages.

3.5: Fault-Injection Components

Fault-Injection components are designed to support testing and evaluation of the dependability of the SWFT components and systems [10]. There are four Fault-Injection components: *Fault-Injection Manager*, *Fault-Injection Receptor*, *Data Collector*, and *Data Analyzer*. The Fault-Injection Manager component provides an interface to the tester for specifying fault-injection parameters such as the location at which the fault should be injected and its duration. Each node has its own Fault-Injection Receptor component for injecting the fault to occur in the node, and a Data Logger to log events and data. The Data Analyzer component is for off-line analysis of the fault-injection data.

4: Graphical Tools

The other major ingredient constituting the integrated environment of ReSoFT is a set of graphical tools. The *SWFT System Builder* facilitates creation of SWFT systems from the reusable components. The *SWFT System Monitor* provides status information during system execution. And the *Fault-Injection Manager* allows dependability evaluation of the SWFT components and systems through fault-injection testing.

The graphical user interfaces (GUIs) of the tools were implemented using the Tcl scripting language and the Tk toolkit [13]. They provide a fast prototyping environment for creating GUIs under X Windows. However, when fast periodic graphics updates of the interface is needed, such as the case in the SWFT System Monitor, Tcl can sometimes be too slow because it is an interpretive language.

4.1: SWFT System Builder

The conventional approach to support reuse of software components is to provide a repository management system which has the capability of representing, browsing, navigating, and retrieving components from the library [8]. For a specific and small reuse domain such as software fault tolerance it will be more effective to use the application generator approach [6]. In such an approach, specifications are translated into application programs. In ReSoFT, the SWFT System Builder provides a graphical form for the user to input information about the SWFT system and application. Figure 3 shows the graphical form where the user selected the DRB scheme. The dynamic form then asks for the number of alternatives and number of tryblocks in the application. After that, the user will be asked to input, for each tryblock, the name of the procedure for the primary module, alternate module, acceptance test, checkpoint establishment routine, checkpoint restoration routine, and finally the timeout period of the application modules. These procedures are specific to the application and are to be supplied by the user. The information is then used to retrieve the reusable SWFT components which are then compiled with the supplied procedures to create the specified system.

4.2: SWFT System Monitor

The SWFT System Monitor allows the user to observe the execution of the SWFT systems. Figure 4 is a snapshot of the System Monitor for the DRB scheme. During fault-free operation the Monitor highlights the component which is being executed and shows the data flow of the system. In this snapshot, it shows the DRB active node and shadow node are executing the primary and alternate modules, respectively. When a fault occurs, the Monitor shows

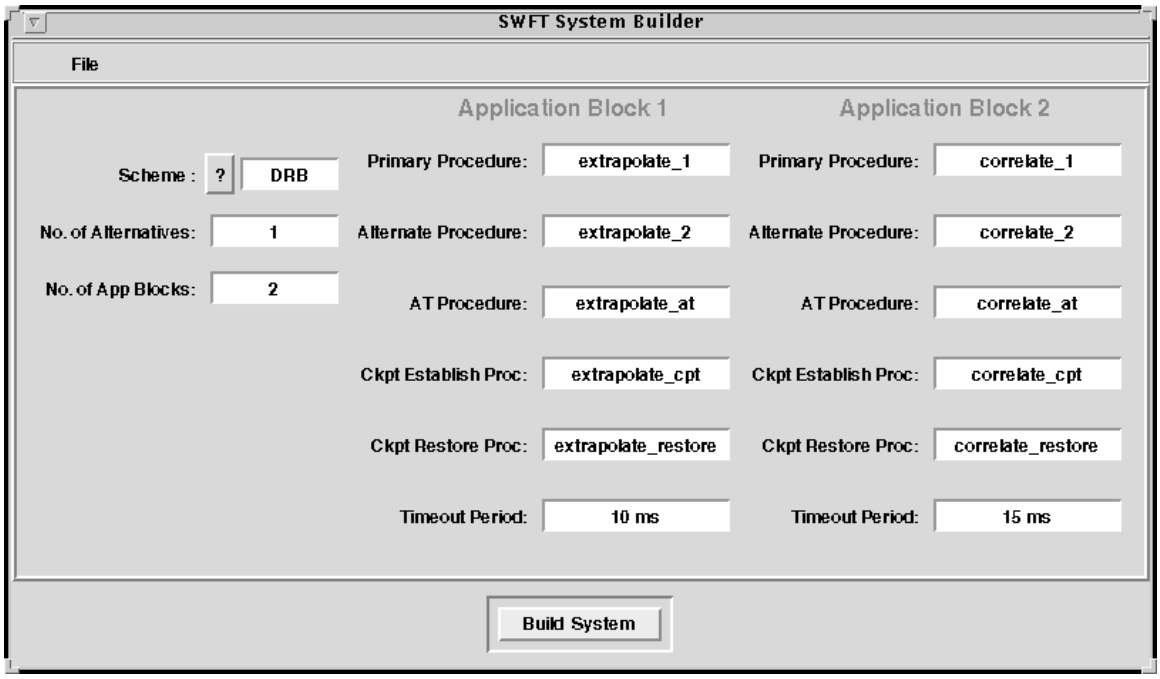

Figure 3. SWFT System Builder

the location and type of the fault, and how it was handled by the SWFT system. The circle in the upper left corner of each node blinks to show the heartbeats, and thus its physical health, of the node. The lower part of the GUI shows the global view of the testbed. It highlights the node or network when its failure is detected.

4.3: Fault-Injection Manager

The Fault-Injection Manager provides an interface to the user for specifying fault-injection experiments, as shown in Figure 5. Each experiment consists of a list of fault sequences. A fault sequence specifies the location where the fault should be injected, the fault type, the starting time of injection, how often it should be periodically injected if it is a transient fault, and duration of the experiment. Once the fault sequences are specified and the experiment is initiated, the FI Manager parses the sequences to schedule the experiment. When the time arrives for a fault to be injected, it sends a fault-injection command to the Fault-Injection Receptor of specified component which in turn triggers the Fault-Injection Activator of the particular fault type to inject the fault. The FI Manager also monitors the results of the injection. In case the injected fault causes a node to crash the FI Manager restarts the node so that experiment can continue. An additional task of the FI Manager is to log all the events occurred and collect failure data.

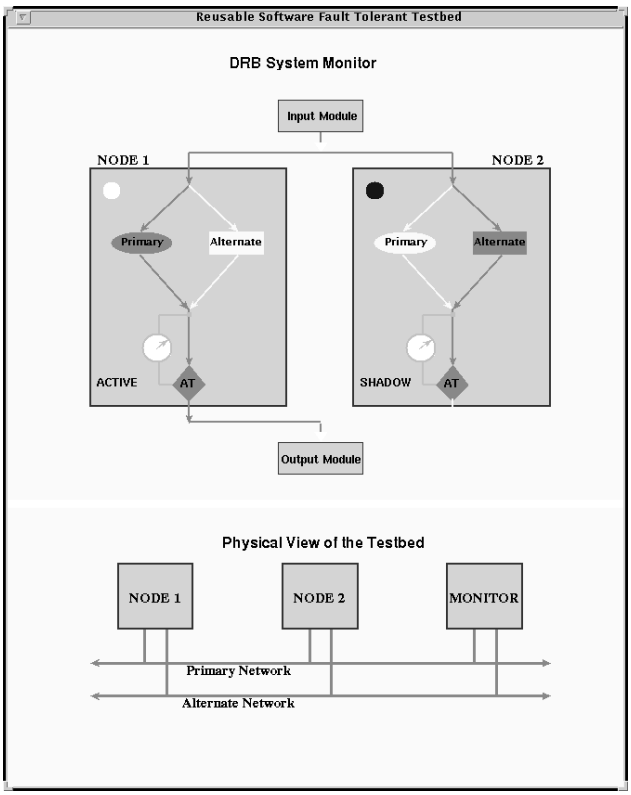

Figure 4. SWFT System Monitor for Distributed Recovery Blocks

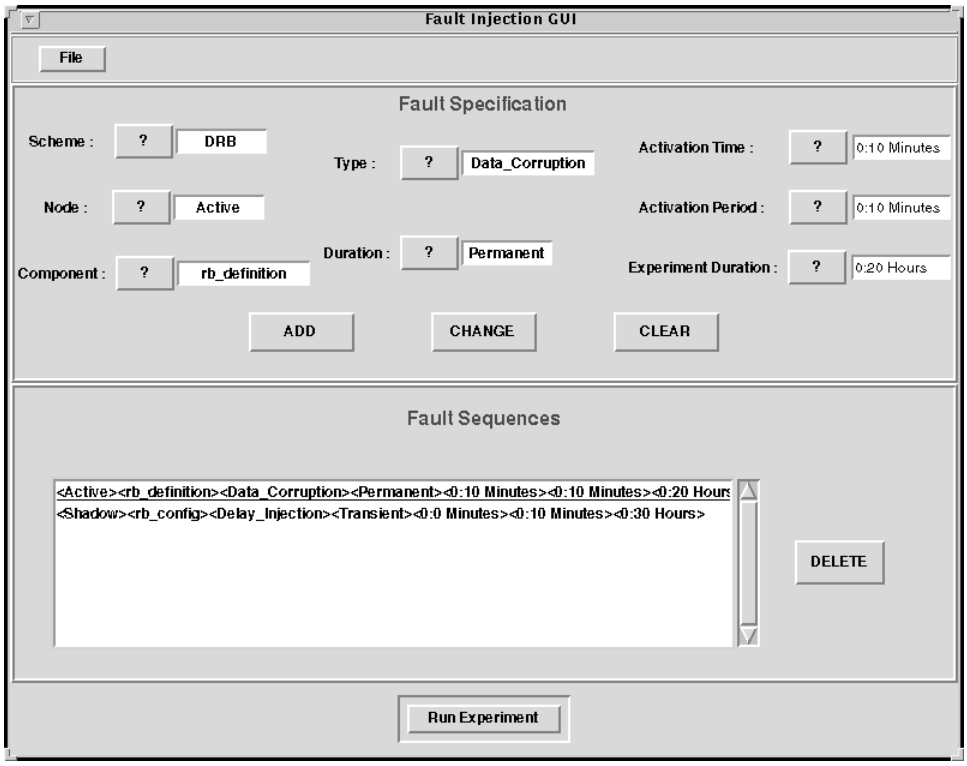

Figure 5. Fault-Injection Manager

5: Distributed Recovery Blocks Demonstration

Figure 6 depicts the implementation of the Distributed Recovery Block (DRB) scheme [11] using the reusable SWFT components. The reusable components are implemented in three types of objects:

- *Active objects*: An active object is an object which possesses its own execution thread(s). Additionally, an active object may also provide services to other objects.
- *Passive objects*: Passive objects do not have their own execution threads but merely act as service providers. Passive objects are stateless and do not maintain any persistent data.
- *Shared-data objects*: Shared-data objects maintain data stores to which multiple objects may have mutually-exclusive access.

These object types are implemented by mapping them to the following Ada constructs. An active object is implemented as an Ada package with a task for each of its execution threads. While a passive object is implemented as an Ada package which does not contain any task. Finally, a shared-data object is implemented using Ada protected record type, thereby the Ada runtime system will guarantee mutually exclusive accesses to the shared-data.

In the DRB implementation, the *DRB Executive*, *Heartbeat-Generator*, and *Try Block* are active components, the *Checkpoint* and *Acceptance Test* are passive components, and the *RB Data-Store* is the only shared-data object.

To demonstrate effective reuse, a sequential recovery block and retry block system are also implemented from the reusable components. We have found that only the *executive* component is needed to be replaced for the particular SWFT scheme. For retry block, a *data re-expression* component is also added to generate a diverse data input.

6: Conclusions

The paper presented an integrated testbed environment for developing and evaluating software fault tolerance systems. The testbed is built on an open architecture and uses standard off-the-shelf hardware and software, thus making it easily available and resilient to technological changes. The environment comprises a library of reusable components and a set of graphical tools. The reusable SWFT components can be used to create a wide variety of SWFT systems, and the interface components support seamless integration of applications to those systems. The set of graphical tools facilitates the creation of SWFT systems

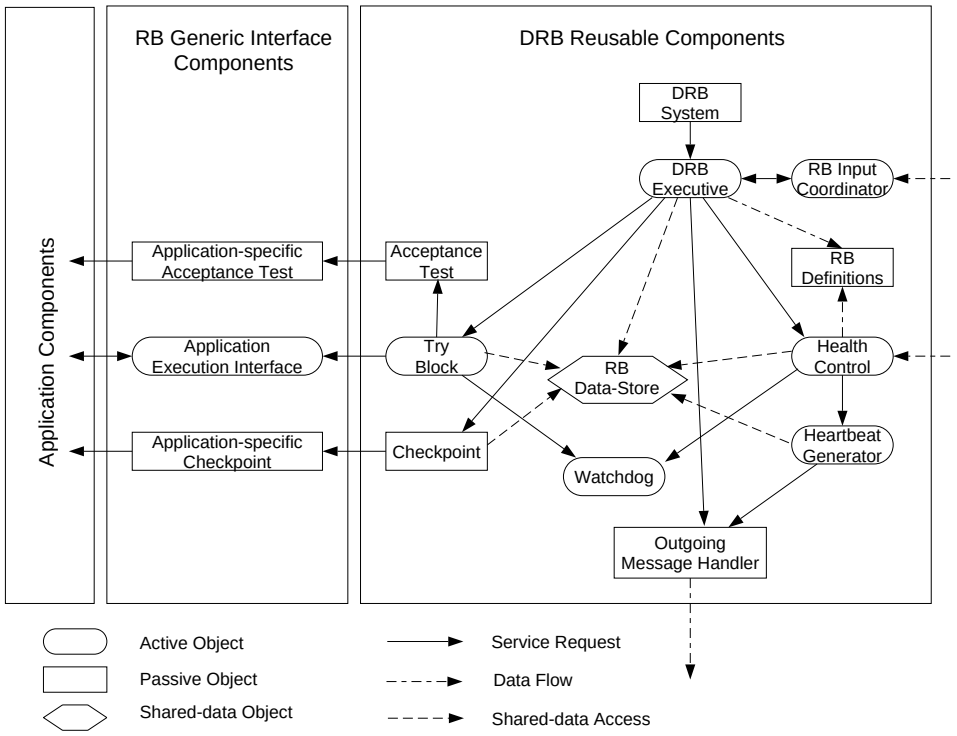

Figure 6. Components-Based Implementation of the DRB Scheme

from the reusable components, monitoring the execution of the resulting systems, and evaluation of their dependability through fault-injection testing.

References

[1] P. E. Ammann and J. C. Knight. Data diversity: An approach to software fault tolerance. *IEEE Trans. Comput.*, pages 418–425, Apr. 1988.

[2] T. Anderson, P. A. Barrett, D. N. Halliwell, and M. R. Moulding. Software fault tolerance: An evaluation. *IEEE Trans. Softw. Eng.*, SE-11(12):1502–1510, Dec. 1985.

[3] A. Avižienis. Design diversity — the challenge for the eighties. In *Digest of the 12th Annual International Symposium on Fault-Tolerant Computing*, pages 44–45, June 1982.

[4] P. G. Bishop, D. G. Esp, M. Barnes, P. Humphreys, and G. Dahll. PODS — a project of diverse software. *IEEE Trans. Softw. Eng.*, SE-12(9):929–940, Sept. 1986.

[5] G. Booch. *Object Oriented Analysis and Design with Applications*. Benjamin/Cummings, Redwood City, CA, 2nd edition, 1994.

[6] J. C. Cleaveland. Building application generators. *IEEE Software*, pages 25–33, July 1988.

[7] D. E. Eckhardt and L. D. Lee. A theoretical basis for the analysis of multiversion software subject to coincident errors. *IEEE Trans. Softw. Eng.*, SE-11(12):1511–1517, Dec. 1985.

[8] W. B. Frakes and P. B. Gandel. Classification, storage and retrieval of reusable components. In *Proc. SIGIR'89*, pages 251–254, Cambridge, MA, June 1989.

[9] ISO/IEC-8652:1995. *Ada Reference Manual: Language and Standard Libraries*. International Organization for Standardization and International Electrotechnical Commission, Jan. 1995.

[10] R. K. Iyer and D. Tang. Experimental analysis of computer system dependability. In D. Pradhan, editor, *Fault-Tolerant Computing: Theory and Techniques*. Prentice Hall, 2nd edition, 1995.

[11] K. H. Kim and H. O. Welch. Distributed execution of recovery blocks: An approach for uniform treatment of hardware and software faults in real-time applications. *IEEE Trans. Comput.*, 38(5):626–636, May 1989.

[12] J. C. Knight and N. G. Leverson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Trans. Softw. Eng.*, SE-12(1):96–109, Jan. 1986.

[13] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, MA, 1994.

[14] E. H. Shokri and K. S. Tso. Ada95 object-oriented and real-time support for development of software fault tolerance reusable components. In *Proceedings of Second International Workshop on Object-oriented Real-time Dependable Systems (WORDS'96)*, Laguna Beach, CA, Feb. 1996.

[15] K. S. Tso, E. H. Shokri, A. T. Tai, and R. J. Dziegiel, Jr. A reuse framework for software fault tolerance. In *Proceedings of AIAA Computing in Aerospace 10 Conference*, pages 490–500, San Antonio, TX, Mar. 1995.