ReSoFT: A Reusable Testbed for Development and Evaluation of Software Fault-Tolerant Systems

Kam S. Tso
IA Tech, Inc.
Los Angeles, CA 90024
Email: k.tso@ieee.org

Eltefaat H. Shokri
SoHaR Incorporated
Beverly Hills, CA 90211
Email: shokri@sohar.com

Roger J. Dziegiel, Jr.
Rome Laboratory/C3CB
Rome, NY 13441
Email: dziegielr@rl.af.mil

Abstract

The Reusable Software Fault Tolerance Testbed (ReSoFT) was developed to facilitate the development and evaluation of high-assurance systems that require tolerance of both hardware and software faults. Central to ReSoFT is a library of reusable software components for the construction of target systems that utilize a wide variety of software fault tolerance (SWFT) techniques. The reusable components include 1) SWFT Executive components, 2) SWFT Support components, 3) Application Interface components, 4) Fault-injection components, and 5) Communications components. A set of graphical tools are also provided for users to build, monitor, and test the SWFT systems.

The ReSoFT testbed was developed on a network of Sun workstations running the Solaris operating systems. The workstations are connected with dual-redundant networks. Object-oriented analysis and design based on the Booch Method had been used to develop the reusable components. The components are implemented in Ada 95 to take advantage of its new object-oriented and real-time support features. The graphical tools were implemented in Java.

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. Some experiments demonstrated favorable results when SWFT is properly applied [1, 2]. However, a few theoretical and empirical studies raised concerns on the validity of the design diversity approach which conjectures that independent development can minimize the probability of common-mode errors [3, 4]. The need to experiment with software fault tolerance to investigate its effectiveness and the desire to reduce effort in building such experimental SWFT systems motivated the design and development of the Reusable Software Fault Tolerance Testbed (ReSoFT). ReSoFT provides (1) a library of reusable components from which software fault-tolerant systems can be built, and (2) a set of graphical tools facilitating the construction of such systems, monitoring execution status, and their dependability evaluation. Figure 1 shows an overview of the ReSoFT integrated hardware and software environment.

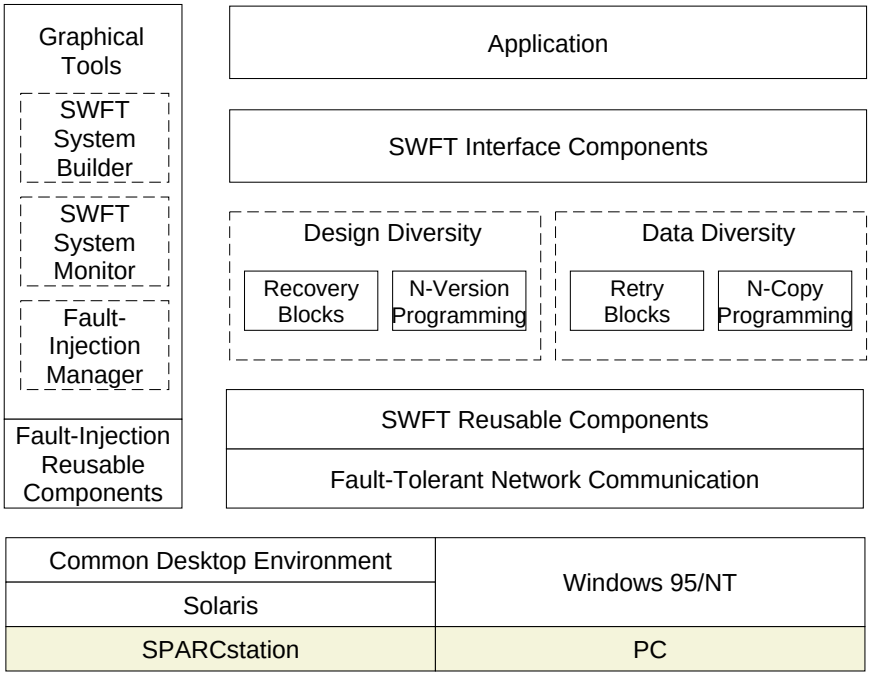

Figure 1: Overview of the ReSoFT Environment

The hardware platform of the testbed, shown in Figure 2, is a network of Sun worksta-

tions running the Solaris operating system. The widespread use of PCs and the availability of 32-bit multitasking Windows 95/NT operating systems have also prompted us to port the ReSoFT environment to the PC/Windows platform. The use of an open architecture and commercial-off-the-shelf hardware and software products guarantee that the testbed can be 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. Therefore, the workstations are connected with dual-redundant Ethernet networks for tolerating network hardware and software faults. One network uses twisted pair while the other uses coaxial cable for physical connection.

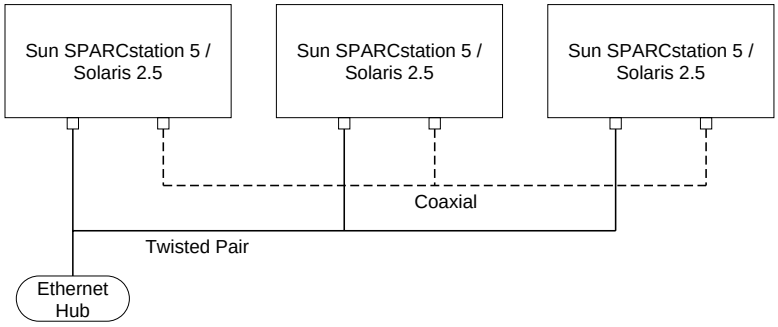

Figure 2: Hardware Configuration of ReSoFT

As shown in Figure 1, the ReSoFT software environment consists of a number of layers. The bottom layer is the network communication which provides fault-tolerant communication over the redundant networks. The next layer includes the reusable SWFT components from which SWFT systems are built. Currently, the ReSoFT testbed provides reusable SWFT components for the following two diversity paradigms: (1) *design diversity* which makes use of diverse techniques in design and implementation as exemplified by recovery blocks [5] and N-version programming [6], and (2) *data diversity* which makes use of diverse data inputs as exemplified by retry blocks and N-copy programming [7]. In addition to the reusable components, the ReSoFT environment also provides a set of graphical tools facilitating the building, monitoring, and testing of SWFT systems.

2 ReSoFT Reusable Components

An in-depth object-oriented analysis of the well-established SWFT schemes has been conducted using the Booch method [8] for identifying the 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 separated from the generic reusable components. However, application-specific functions are derived from generic classes using the inheritance mechanism when the application is integrated with the SWFT system. The result of the analysis is a reuse framework for software fault tolerance and the details were reported in [9].

According to their functionality, identified components are 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 [10]. 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 supported software reuse and real-time processing in implementing the SWFT reusable components can be found in [11].

2.1 SWFT Executive Components

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

2.2 SWFT Support Components

Comparing existing SWFT schemes, we find several generic fault tolerance related components that are commonly used. We designate them as SWFT support components. Seven types of reusable SWFT Support components have been identified by domain analysis: *Try Block*, *Acceptance Test*, *Decision Algorithm*, *Heartbeat*, *Watchdog*, *Checkpointing*, and *Data Re-expression* components. Detailed description of these components can be found in [9].

2.3 SWFT Interface Components

The ReSoFT software components were 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 the object-oriented language has been used to separate application-specific concepts from application-independent ones. The SWFT Interface components are designed as abstract classes and the application designer creates subclasses for adding the application-specific functions. The *Application-Specific Acceptance Test*, *Application-Specific Decision Algorithm*, *Application-Specific Data Re-expression*, and the *Application-Specific Checkpointing* components belong to interface components.

2.4 Fault-Tolerant Network Communication Components

The Network Communication components provide 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 inter-node 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 of these two duplicated messages 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 recognizing the loss of several consecutive messages.

2.5 Fault-Injection Components

Fault-Injection components are designed to support testing and evaluation of the dependability of the SWFT components and systems. 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.

3 ReSoFT Graphical Tools

The other major ingredient constituting the integrated environment of ReSoFT is a set of graphical tools to facilitate the building, monitoring, and fault-injection of SWFT systems. Figure 3 shows the tools being used in monitoring the execution and conducting fault-injection testing of the N-Version Programming system.

The tools are launched by pressing the corresponding image button of the SWFT Toolset Launcher. Currently four tools are available: The *SWFT System Builder* facilitates the construction of SWFT systems from the reusable components. The *SWFT System Monitor* provides status information during system execution. The *Fault-Injection Manager* allows dependability evaluation of the SWFT components and systems through fault-injection testing. And the *Application Viewer* which is an application-specific tool for viewing the results of the application.

The tools were implemented in Java [12] which allows them to be run on different platforms. They can be run as either Java applets or Java applications. As Java applets, they

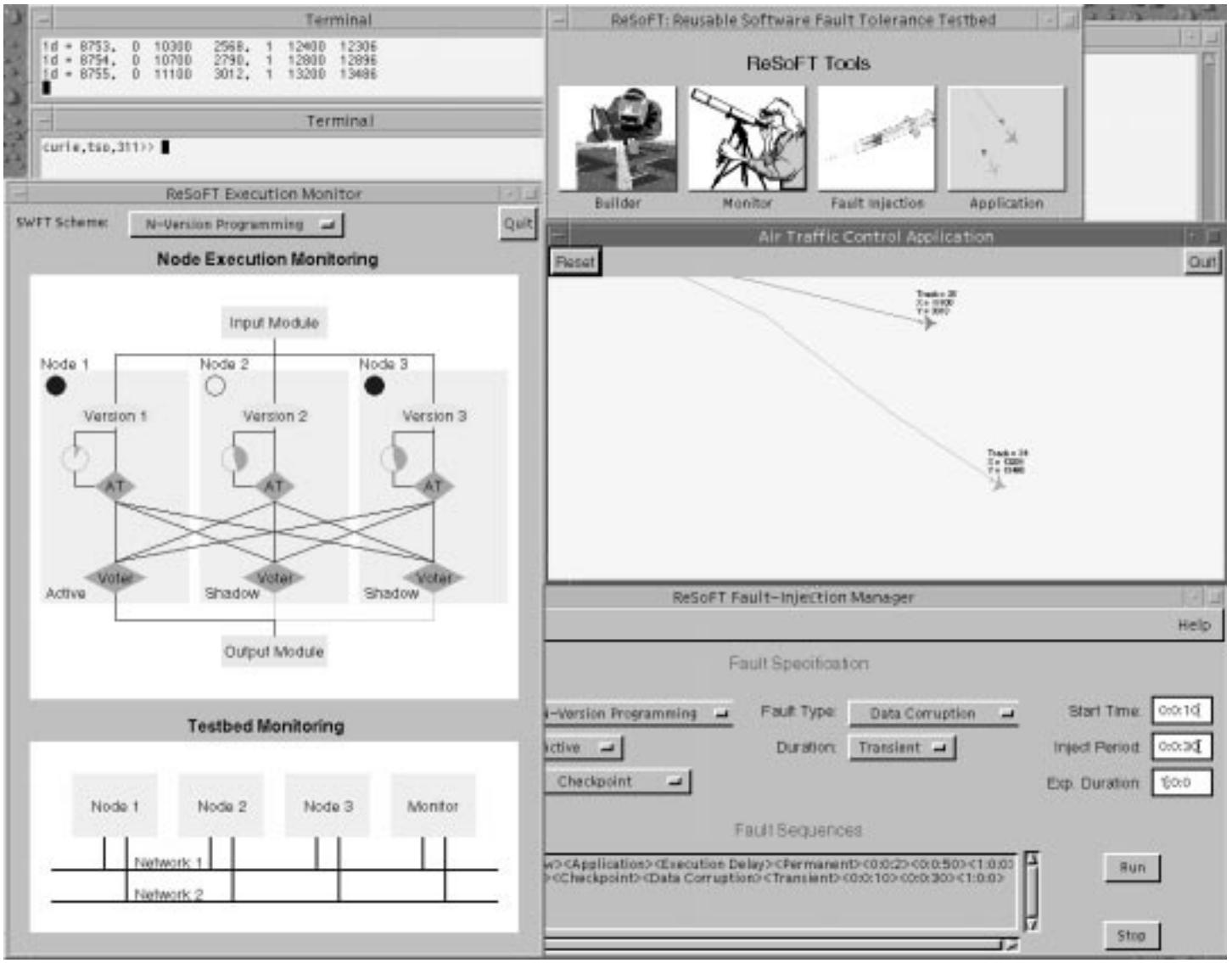

Figure 3: Snapshot of the ReSoFT Toolset during Execution of an NVP System

can be downloaded from the World Wide Web [1] and run on any Web browser that supports Java, such as Netscape Navigator or Internet Explorer. Because of security restriction, Java applets cannot access local files or make network connection to hosts other than the Web server. As a result, the ReSoFT tools, when running as applets, will not be fully functioning as to retrieving reusable components or receiving messages from other nodes.

3.1 SWFT System Builder

The conventional approach to support reuse of software components is to provide a repository management system with the functionalities of representing, browsing, navigating, and retrieving components from the library [13]. We found that, for a specific and small reuse domain such as software fault tolerance, effective reuse can be better achieved with the application generator approach [14]. In such an approach, specifications are translated into application programs. In ReSoFT, the SWFT System Builder provides a graphical interface for the user to input information about the application and the selected SWFT scheme. Figure 4 shows the graphical user interface where the user selected the N-Version Programming (NVP) scheme.

Prior to using the System Builder, the user will need to prepare the application modules (the primary and alternate modules for Recovery Blocks, the various versions for N-Version Programming) and also implements the SWFT interface components specific to the application. The user can then input the relevant information to the System Builder for the target system, including the selection of the SWFT scheme, locations of the Component Repository, target system, and target application, timing information, host names where the system is to be executed, and whether the hosts are connected by single or dual-redundant networks. After all these information are specified, the user pushes the **Build** button and the System Builder will retrieve the necessary components from the repository and compile them with the application modules and interface components to create the target software fault-tolerant

[1] The ReSoFT Toolset can be accessed at `http://www.sohar.com/~tso/swft-tools/index.html`

systems. The Builder also creates scripts with the needed command-line arguments to run the systems.

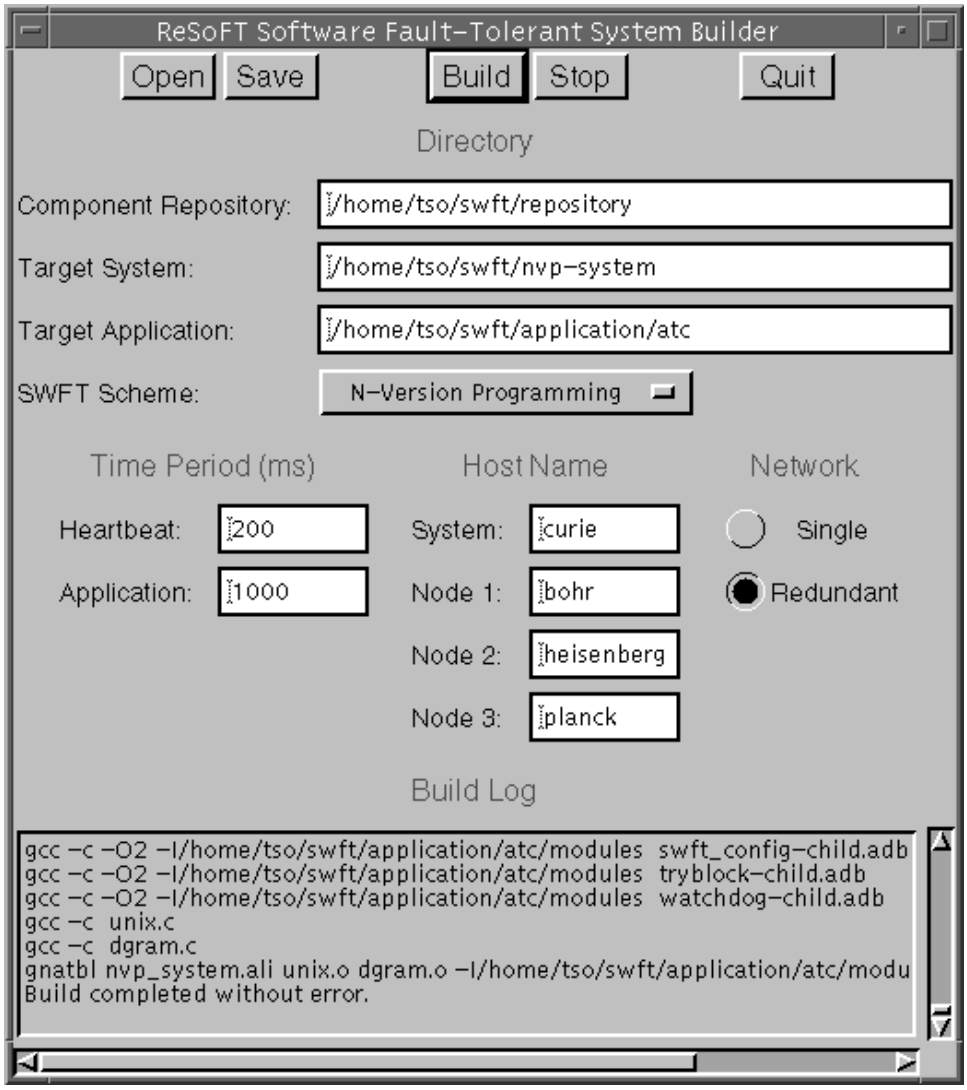

Figure 4: SWFT System Builder

3.2 SWFT System Monitor

The SWFT System Monitor allows the user to observe the current status and occurrence of events during the execution of the SWFT systems. The Monitor works for all five currently implemented SWFT schemes. The user can use the SWFT Scheme choice menu to select the monitoring of the SWFT scheme. The Monitor also switches to the scheme specified in

the messages from the SWFT nodes.

Figure 5 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 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.

3.3 Fault-Injection Manager

The Fault-Injection (FI) Manager provides an interface to the user for specifying fault-injection experiments, as shown in Figure 6. 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 the experiment can continue. An additional task of the FI Manager is to log all the events occurred and collect failure data.

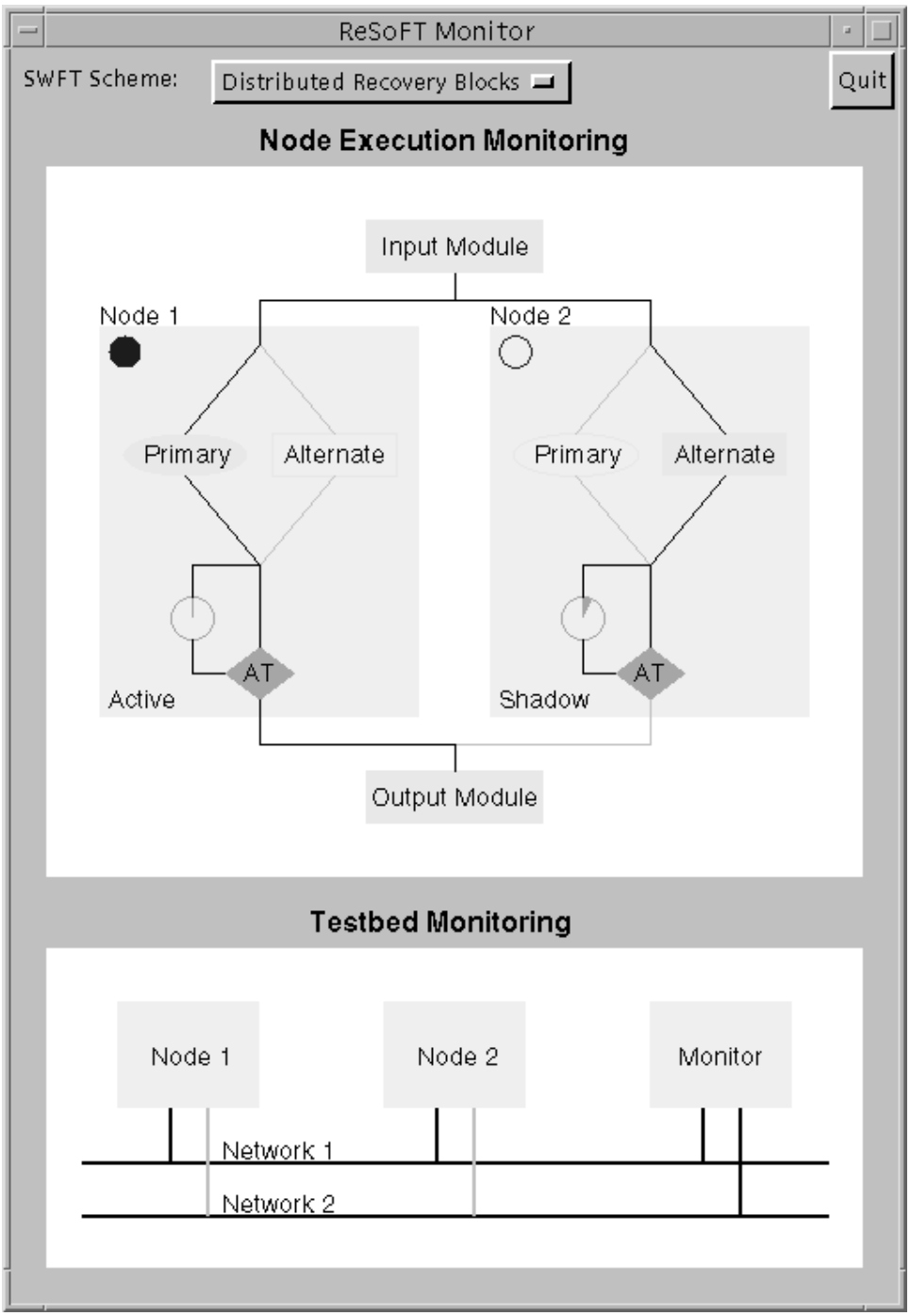

Figure 5: SWFT System Monitor for Distributed Recovery Blocks

4 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

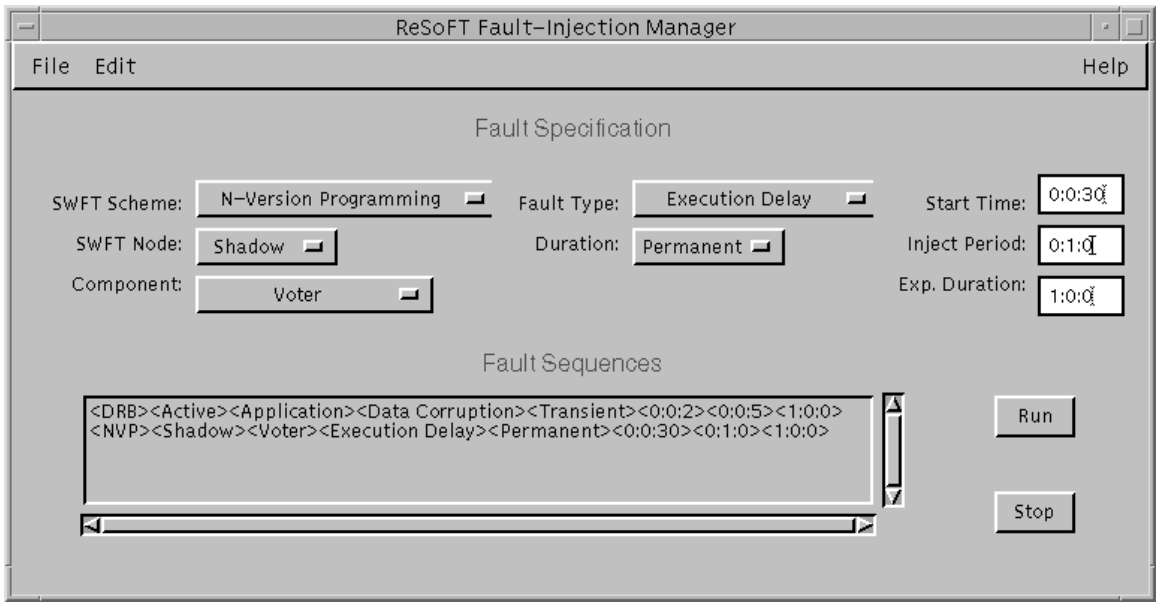

Figure 6: Fault-Injection Manager

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

We believe that ReSoFT can be used by academia and research laboratories in a number of ways.

- New SWFT schemes can be prototyped and evaluated quickly by reusing the available reusable components and tools.
- SWFT applications using the existing SWFT schemes can be constructed automatically from the reusable components using the System Builder.
- SWFT experiments can be conducted on the testbed using the fault-injection and logging facilities to collect failure data.

References

[1] T. Anderson, P. A. Barrett, D. N. Halliwell, and M. R. Moulding, "Software fault tolerance: An evaluation," *IEEE Trans. Software Engineering*, vol. SE-11, pp. 1502–1510, Dec. 1985.

[2] P. G. Bishop, D. G. Esp, M. Barnes, P. Humphreys, and G. Dahll, "PODS — a project of diverse software," *IEEE Trans. Software Engineering*, vol. SE-12, pp. 929–940, Sept. 1986.

[3] D. E. Eckhardt and L. D. Lee, "A theoretical basis for the analysis of multiversion software subject to coincident errors," *IEEE Trans. Software Engineering*, vol. SE-11, pp. 1511–1517, Dec. 1985.

[4] J. C. Knight and N. G. Leverson, "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Trans. Software Engineering*, vol. SE-12, pp. 96–109, Jan. 1986.

[5] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Engineering*, vol. SE-1, pp. 220–232, June 1975.

[6] A. Avižienis, "The N-Version approach to fault-tolerant software," *IEEE Trans. Software Engineering*, vol. SE-11, pp. 1491–1501, Dec. 1985.

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

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

[9] 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*, (San Antonio, TX), pp. 490–500, Mar. 1995.

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

[11] 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.

[12] M. A. Hamilton, "Java and the shift to net-centric computing," *IEEE Computer*, vol. 29, pp. 31–39, Aug. 1996.

[13] W. B. Frakes and P. B. Gandel, "Classification, storage and retrieval of reusable components," in *Proc. SIGIR'89*, (Cambridge, MA), pp. 251–254, June 1989.

[14] J. C. Cleaveland, "Building application generators," *IEEE Software*, pp. 25–33, July 1988.