Pacific Rim International Symposium of Fault-Tolerant Systems
Newport Beach, CA, pp. 98-103, Dec. 1995.

ReSoFT: A Reusable Software Fault Tolerance Testbed

Kam S. Tso and Eltefaat H. Shokri
SoHaR Incorporated
Beverly Hills, CA 90211

Abstract

The need to experiment with software fault tolerance (SWFT) for investigating its effectiveness and the desire to reduce effort in creating those experimental SWFT systems have motivated the development of the Reusable Software Fault Tolerance Testbed (ReSoFT). The testbed is built based on an open architecture and uses standard off-the-shelf hardware and software, thus making it easily available and resilient to technological changes. ReSoFT provides a library of reusable components for the users to create systems utilizing a wide variety of SWFT techniques. In addition, interface components that support the seamless integration of applications into the SWFT systems, and fault-injection components that support dependability testing and evaluation are also provided. The paper presents a SWFT reuse framework that resulted from an analysis of the SWFT domain, the design of the SWFT reusable components using the object-oriented method, and configuration of the testbed supporting the creation, execution, and evaluation of the SWFT systems.

1 Introduction

The interest in software fault tolerance has been growing as software faults have become the major contributor to system failures, and there has been a lack of breakthroughs in software practices to improve the situation. Although 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 [1, 2]. However, some theoretical and empirical studies raised concerns on the validity of the design diversity approach which conjectures that the independence of the development process will minimize the probability of similar errors [3, 4].

Experimentation in SWFT techniques is hindered by the difficulty of incorporating them in complex systems. Although several SWFT techniques have been proposed, only a few research efforts have been reported to establish guidelines for cost-effective implementation of SWFT techniques. Moreover, the majority of SWFT techniques are not transparent to the application developer, meaning that the developer has to implement the elements 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 for investigating its effectiveness and the desire to reduce repetitive effort in creating those experimental SWFT systems have motivated the U.S. Air Force to sponsor the development of the Reusable Software Fault Tolerance Testbed (ReSoFT). ReSoFT has been designed with the following objectives in mind.

1. *Configurability*: The ability to configure a wide range of experimental systems utilizing existing and new SWFT techniques.
2. *Integrability*: The ability to integrate and execute a wide range of applications, with minimum modifications, on the SWFT systems.
3. *Availability*: The hardware and software required to develop the testbed should be readily available in most national laboratories and universities.
4. *Usability*: tools should be available to facilitate the experimentation and evaluation of software fault tolerance.

Our approach to accomplish these objectives in ReSoFT is to develop a reuse framework which captures the essential functionality of the software fault tolerance domain. Common operations within the framework are then identified as reusable components and are used as building blocks for implementing a wide variety of SWFT systems. Interface components that support the seamless integration of applications into the SWFT systems, and fault-injection components that support dependability testing and evaluation, are provided in addition to the SWFT components. Users of the testbed interact with a graphical user interface to create standard SWFT systems by means of pre-defined templates or custom SWFT systems by assembling the available reusable components. The testbed is built on a network of Unix workstations which consists of widely available, commercial-off-the-shelf, standard hardware and software products.

Several testbeds have been developed to study the issues of reliable distributed system design [5] and hardware fault tolerance [6], but few of them addressed software fault tolerance. One of the early SWFT testbeds is DEDIX, developed at UCLA for N-version software, was implemented on a network of VAX minicomputers [7]. While DEDIX accomplished its goal to support experimentation with N-Version software, it was developed on a research distributed operating system and became obsolete when the operating system was removed from the network. Another SWFT testbed supporting the execution of Distributed Recovery Blocks was developed at UCI [8]. The system supports the investigating of the real-time issues because it was implemented using a tightly coupled interconnection network of custom processor boards. A major advantage of ReSoFT as an experimental testbed over the previous ones is that it provides a component-based environment to allow users to create their specific SWFT systems from the existing reusable components. The users can also modify existing components or add new ones to build custom SWFT systems according to their need. Recently, an interesting reuse effort by Huang and Kintala [9] developed a library of C functions to detect and recover from failures such as system hang and crash.

The rest of the paper is organized as follows: Section 2 describes the hardware and software of the testbed. The results of the domain analysis identifying the SWFT reusable components are presented in Section 3. Section 4 describes the design of the components. Section 5 concludes the paper.

2 Software and Hardware Configurations

The ReSoFT testbed is developed based on an open architecture employing commercial-off-the-shelf hardware and software products. The testbed consists of a collection of computing nodes made of Sun SPARCstation 5 workstations running the standard Solaris 2.4 operating system. The workstations are using the TCP/IP protocol.

Implementing ReSoFT on a network of Unix workstations has the following advantages:

- *Economical.* It minimizes the cost of running the testbed because major universities and research laboratories have a network of Unix workstations in place.

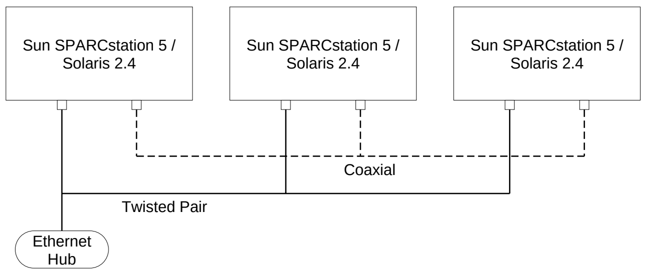
Sun SPARCstation 5 / Solaris 2.4
Sun SPARCstation 5 / Solaris 2.4
Sun SPARCstation 5 / Solaris 2.4
Coaxial
Twisted Pair
Ethernet Hub

Figure 1: Hardware Configuration of ReSoFT

- *Resilient to technological change.* Processor and operating system upgrade has almost become an annual event. Having the testbed implemented on a standard open architecture using commercial-off-the-shelf hardware and software products facilitates upgrading.

- *Ease of use.* Use of the popular Unix environment running on commodity workstations avoids users having to learn and maintain the special hardware and software elements required by a custom-built system.

Communication among the ReSoFT nodes utilizes the standard TCP/IP protocol over the Ethernet. As the fault-tolerant detection and recovery mechanisms rely highly on the reliable communication among the nodes, the network becomes the single point of failure. In order to tolerate network hardware and software faults, the workstations are connected by dual-redundant Ethernet networks. One uses twisted pair (IEEE 802.3 10BaseT) while the other uses coaxial cable (IEEE 802.3 10Base2) for physical connection. Network communication faults are tolerated by sending identical messages over both networks.

The software components of ReSoFT are developed and executed on top of the Sun Solaris 2.4 operating system. We have chosen Solaris over the more mature SunOS because the former can provide "soft" real-time performance, meaning that deadlines will be met most of the time. The real-time capability is important because the testbed needs bounded response times. However, occasional missing of deadlines does not affect the testbed because the SWFT systems are designed to handle timing faults, along with other types of faults. Solaris achieves the soft real-time performance with an efficient, preemptive, multi-threaded kernel and a real-time scheduling policy. ReSoFT also makes use of the available POSIX 1003.1b thread library for improved response times.

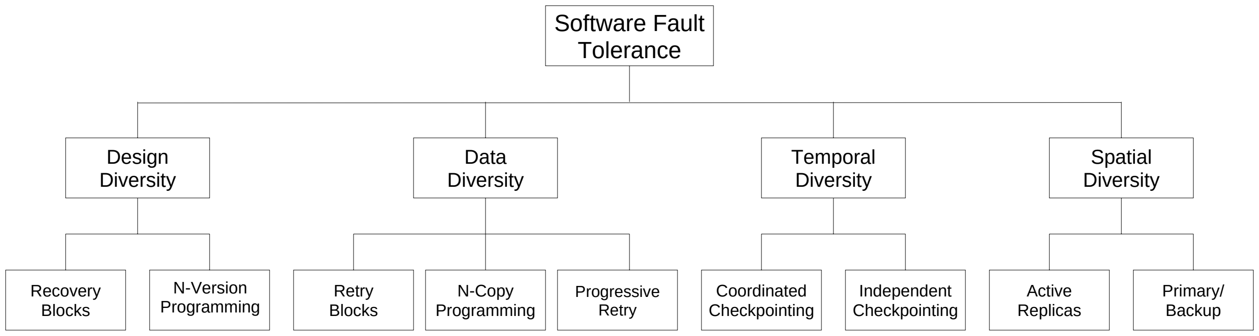

Figure 2: Top Level Taxonomy of the Software Fault Tolerance Domain

3 Taxonomy of SWFT

Domain analysis is an examination of a specific problem domain aimed at identifying common operations, objects, and structures which are candidates for components of the system to be designed. Our domain analysis is based on the study of existing systems and published reports and articles. Figure 2 summarizes the results of the analysis organized as a taxonomy of the software fault tolerance techniques.

3.1 Design Diversity Strategy

Witnessing the level of complexity in emerging applications, one cannot exclusively rely on rigorous design and validation techniques to obtain sufficiently reliable software. It is sensible to seek means for enhancing software systems to cope with residual design errors/inadequacies that may appear at unpredictable instances during system operation. A well-established class of techniques for tolerating residual software design failures is to provide multiple designs (or so-called versions) for a given software specification. An executive component may then be dedicated to executing the prepared versions and providing a correct output based on the results produced by the versions, even in the presence of failures. The underlying assumption here is that the diverse versions do not contain the same set of errors. The Recovery Block [10] and N-Version Programming [11] are among the well-established design diversity schemes.

3.2 Data Diversity Strategy

Extensive testing of a software system typically identifies the software errors that manifest themselves most frequently. As a result, most of the remaining software errors are of the types which occur only in very special and intuitively-rare scenarios that are overlooked in the testing and validation phases. The data diversity strategy is based on the partially validated assumption that if the data set is re-expressed (i.e., it is either expressed in some other form, or transformed into an equivalent data set), the software that failed with the original data set, does not fail with the re-expressed data set. Progressive Retry [12], Retry Block and N-Copy Programming [13] schemes belong to this strategy.

3.3 Temporal Diversity Strategy

It is well known that most of hardware/software failures in computer systems are soft (also called *Heisenbugs*) [14], meaning that if the failed scenario is repeated (at different time or on different hardware), the failure disappears and the system will succeed to produce acceptable results. This class of failures is typically created by transient hardware failures or unanticipated sequences of external events. The temporal diversity strategy suggests that to recover from a soft failure, the failed scenario should be repeated immediately after the failure is detected. Checkpointing and rollback schemes [12, 15] utilize temporal diversity for system recovery.

3.4 Spatial Diversity Strategy

The logic supporting spatial diversity is based on the assumption that if the failed scenario is repeated in different hardware, most failures will not be repeated [14]. Under this scheme, the application software is executed in multiple hardware units and an executive component specifies the correct output of the system using the outputs of hardware units. Primary/backup protocols [16] employ spatial diversity to tolerate hardware/software failures.

4 ReSoFT Software Architecture

ReSoFT consists of several software layers: (i) Ada binding to support network communication, (ii) reusable SWFT components, (iii) SWFT schemes employing the reusable components, (iv) application layer, and (v) fault-injection layer. A detailed discussion of this layered structuring of ReSoFT architecture can be found in [17].

To identify reusable components, we conducted an object-oriented analysis of the well-established SWFT schemes using Booch object-oriented method [18]. During this process, care was taken to make sure that (i) the components capture the common functionality of a wide variety of SWFT schemes, and (ii) 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 was reported in [17].

Figure 3 depicts the result of analysis for the Recovery Block scheme. As shown in the diagram, the *Recovery Block Scheme* class uses *Acceptance Test*, *Try Block*, and *Checkpointing Mechanism* classes. The use relationship is denoted by a line ended with an empty circle at the user side. The Recovery Block Scheme also has an aggregation relationship (denoted by a line ended with a solid circle at the owner side) with the *Executive* class. On the other hand, the Executive class itself is an abstract class (denoted by an inverted triangle enclosing an A) with *Single-Process Recovery* and *Concurrent-Processes Recovery* classes as its subclasses (the subclass relationship is denoted by an arrowed line). Single-Process Recovery, in turn, has two subclasses: *SRB Executive* and *DRB Executive* classes. Moreover, *PTC Executive* and *Conversation Executive* classes (representing two known approaches for the recovery of concurrent processes) are concrete subclasses of the Concurrent-Processes Recovery class. The Try Block class has two subclasses: *Primary* and *Alternate*, and *Acceptance Test* is the superclass of *Timing Checks* and *Reasonableness Checks* classes. Finally, *Checkpointing Mechanism* owns three subclasses: *Recovery Cache*, *Checkpoint*, and *Audit Trail* classes.

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

4.1 SWFT Executive Components

The SWFT executive components are responsible for managing orderly execution of the SWFT schemes which may 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 of the scheme, there can be one or more executive components.

4.2 SWFT-specific 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 support components have been identified during domain analysis: *Try Block*, *Acceptance Test*, *Voter*, *Heartbeat*, *Watchdog*, *Checkpointing*, and *Data Re-expression* components. The functions of most of these components have been formalized in the literature. 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.

4.3 SWFT Interface Components

The most important goal in designing SWFT components is their maximum reusability. These components should be designed in the way that they can be reused by any application with minimal changes in their design and/or implementation. However, there are aspects which vary from one application to another. The inheritance mechanism provided by object-oriented methods can be used to separate the concepts which are application-dependent from the concepts which are application-independent. These components can be designed as abstract classes and the application designer can then introduce subclasses for developing the application specific functions. Subclasses of SWFT components which are introduced for application specific purposes are part of the SWFT interface component category.

Moreover, the interface component category may also include components which are introduced to decouple application components from the SWFT components. Serious efforts should be concentrated on minimizing the changes in the reusable components and also localizing the necessary changes only in the interface components. The *Application-Specific Acceptance Test*, *Application-Specific Checkpointing*, and *Application-Specific Outgoing Message Handler* components belong to interface components. Templates for their implementation are provided by ReSoFT.

4.4 Communication Components

The Network Communication component provides an application transparent reliable inter-node communication among the ReSoFT nodes. Message redundancy management is designed to be transparent to the users. It automatically detects the existence of a second network and establishes alternate connection over it. Fault-tolerant communication is achieved by sending duplicated messages by the *sender* node over

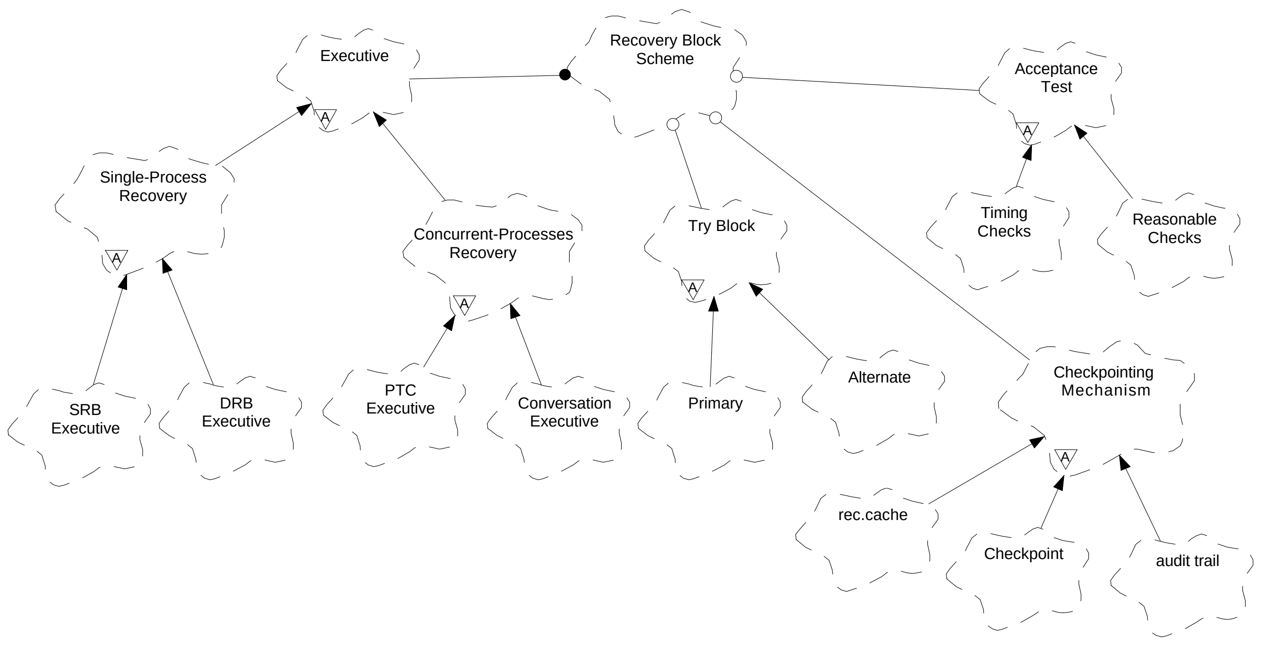

Figure 3: Class Diagram for Recovery Blocks

both connections in a way that the *receiver* recognizes and discards redundant messages. However, if one network fails, it will not be able to deliver messages delivered by the other network. The receiver node diagnoses the problem whenever the number of messages undelivered by a network passes a threshold.

4.5 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.

5 Results and Conclusions

The paper presented the research effort for developing ReSoFT, a software fault tolerance testbed based on reusable components. 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. ReSoFT provides a library of reusable components for users to create systems utilizing a wide variety of SWFT techniques. In addition, interface components that support the seamless integration of applications into the SWFT systems, and fault-injection components that support dependability testing and evaluation are also provided. A graphical user interface has been developed to monitor the status of the testbed as well as SWFT schemes. It will include facilities for the creation of standard SWFT systems by means of pre-defined templates or custom SWFT systems by assembling the available components. The paper presented a SWFT reuse framework that resulted from an analysis of the SWFT domain and the design of the SWFT reusable components using the object-oriented method.

Faults that are currently tolerated by ReSoFT include the following:

- *Timing faults*: This type of faults includes (i) late responses caused by the application and/or system and (ii) infinite delays in the application or system components. Timing faults are detected by the SWFT scheme using a watchdog component.

- *Value (logical) faults*: Value faults in the application (caused by physical or design faults in both the application and system) are detected by an application-specific acceptance test.

- *Node crash faults*: A node may experience a temporary or permanent blackout. Absence of the heartbeat messages of a node is used to detect such faults.
- *Communication faults*: Hardware and software network failures are masked by sending duplicated messages over dual-redundant networks.

Recovery times for all of these failures are in an acceptable range for most applications. The measured worst-case recovery time for any type of failure is about 700 msec when the heartbeat period is set to 200 msec. The recovery time is acceptable considering that the testbed is implemented on a network of workstations running the time-sharing Unix. If needed, the reusable components and the resulting SWFT systems can be ported to special-purpose and more responsive hardware/software platforms.

Acknowledgments

This work was partially supported by Small Business Innovation Research (SBIR) Contract F30602-94-C-0013 from Rome Laboratory, U. S. Air Force.

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] W. W. Chu, K. H. Kim, and W. C. McDonald, "Testbed-based validation of design techniques for reliable distributed real-time systems," *Proc. of the IEEE*, vol. 75, pp. 649–667, May 1987.

[6] A. L. Hopkins, T. B. Smith, and J. H. Lala, "FTMP — a highly reliable fault-tolerant multiprocessor for aircraft," *Proc. of the IEEE*, vol. 66, pp. 1221–1239, Oct. 1978.

[7] A. Avižienis *et al.*, "The UCLA DEDIX system: A distributed testbed for multiple-version software," in *Proc. FTCS-15*, (Ann Arbor, MI), pp. 126–134, June 1985.

[8] K. H. Kim, "An approach to experiment evaluation of real-time fault-tolerant distributed computing systems," *IEEE Trans. Software Engineering*, vol. 15, pp. 715–725, June 1989.

[9] Y. Huang and C. Kintala, "Software implemented fault tolerance: Technologies and experience," in *Proc. FTCS-23*, (Toulouse, France), pp. 2–9, June 1993.

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

[11] A. Avižienis and L. Chen, "On the implementation of N-Version Programming for software fault-tolerance during program execution," in *Proc. of COMPSAC-77*, pp. 149–155, 1977.

[12] Y. M. Wang *et al.*, "Progressive retry for software error recovery in distributed systems," in *Proc. FTCS-23*, (Toulouse, France), pp. 138–144, June 1993.

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

[14] J. Gray, "Why do computers stop and what can be done about it?," in *Proc. 5th Symposium on Reliability in Distributed Software and Database Systems*, (Los Angeles, CA), pp. 3–12, Jan. 1986.

[15] R. Koo and S. Toueg, "Checkpointing and rollback recovery for distributed systems," *IEEE Trans. Software Engineering*, pp. 23–31, Jan. 1987.

[16] D. Powell *et al.*, "The Delta-4 approach to dependability in open distributed systems," in *Proc. FTCS-18*, (Tokyo, Japan), pp. 246–251, June 1988.

[17] K. S. Tso and E. H. Shokri, "Development of reusable ada software fault tolerant components," SBIR Phase II Technical Report, SoHaR Incorporated, Beverly Hills, CA, July 1995.

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