www.icgst.com

Component-Based Recovery Block Technique

O. A. Abulnaja
Department of Computer Science, King Abdulaziz University,
Jeddah, Saudi Arabia,
e-mail: abulnaja@kaau.edu.sa

Abstract

In applications where computer systems are used to manage critical tasks, software faults cannot be tolerated and may lead to catastrophic consequences. Thus, software fault-tolerant techniques must be implemented in systems running such applications. One of the most widely used software fault-tolerant technique is the Recovery Block (RB) technique.

Software components are reusable units that have been independently constructed for building software applications. Component-based software is causing a see change in how software is developed.

This paper introduces a scheme called Component-Based Recovery Block (CBRB) scheme for improving the RB approach reliability. The introduced approach is based on the component technology and the RB technique. Also, the work discusses the expected effect of implementing the proposed technique on system reliability.

Keywords: *Recovery Block, Component Software, Fault-Tolerant Systems.*

1. Introduction

"Between the late 1960s and early 1990s, the software engineering community strove to formalize schemes that would lead to perfectly correct software. Although a noble undertaking at first, it soon become apparent that correct software was, in general, unobtainable. And furthermore, the cost, even if achievable, would be overwhelming" [1]. Because software faults may lead to catastrophic consequences, software fault-tolerant techniques must be implemented in system running critical applications. Various techniques using redundancy have been developed to tolerate software faults and improve systems reliability [2]-[6]. Among these techniques is the well-known Recovery Block (RB) approach.

Software components are reusable units that have been independently constructed (developed by different developers using different algorithms, programming languages, design tools, compilers, implementation techniques, test approaches, hardware, *etc*) for building software applications. The programmers of the units may be with diversity in experience, and training, and widely geographically spread (different companies, or countries).

1.1. Recovery Block (RB) Technique

In the Recovery Block (RB) technique every program or a program module is made up of independently (diversely) designed program modules performing the same task: a primary module (may be the fastest but the most complex) and some alternate modules (may be slower but less complex). First, the primary module is run followed by the *acceptance test* (*AT*). If the module passes the AT, it is considered reliable; otherwise, it is considered faulty and the first alternate module is run. Similarly, if it passes the AT, it is considered reliable; otherwise, it is considered faulty and the next alternate module is run. Modules are run in the order explained above. If any of the modules passes the AT, the output of that module is considered reliable and used. Otherwise, an error message is reported and no output is used [7, 8], see Figure 1.

Several variations of the RB scheme have been proposed in literature [9]-[14]. Also, there have been numerous models of reliability and performance for the RB approach [15]-[18].

1.2. Component-Based Software

The motivation for the component technology came from the limitation of the traditional software applications. A traditional software application usually consists of a single *monolithic* binary file. Once the application is generated, it doesn't changes until the next version is recompiled and shipped. Changes in the operating systems, hardware, and customer desires must all wait for the entire application to be recompiled. The application becomes older and more outdated when it is shipped. This is because the entire software industry rushes on into the future. With the current fast change in the software industry, applications cannot be static after they have shipped. The solution is to break the monolithic application into components as shown in Figure 2. This means that any application can be developed based-on

component, which makes it adaptable to change. When the technology advances, new components can replace the existing ones that build up the application. Figure 3 illustrates that the new improved component D replaced the old component D of Figure 2, hence, the application becomes up to date.

Thus, the application is no longer a static entity destined to be out of date before it ships. Instead the application evolves gracefully over time as new components replace older components. Entirely new applications can be built quickly from existing components. This is due to the reusability feature of the component technology. The component technology can be used to build application that support distribution, multiprocessing, reusability, maintainability, and operating system independent.

There are many software technologies that support components. Among these technologies are *Distributed Component Object Model* (*DCOM*) [19, 20] and *Common Object Request Broker Architecture* (*CORBA*) [20]. Building and deploying software systems across the enterprise is a complex task. DCOM and CORBA provide a powerful framework for accomplishing this. With DCOM and CORBA, we can very easily develop heterogeneous distributed systems.

1.2.1. Distributed Component Object Model (DCOM)

Distributed Component Object Model (DCOM) [19] is the distributed version of the Component Object Model (COM). In DCOM, the *client stub* is referred to as the *proxy* and the *server stub* is referred to as the *stub*. For the sake of simplicity, instead of calling an object function in the same process the object function call could be carried out within some other process possibly on a remote system. Hence there are two calling techniques:

- *In process* call: the calling technique loads the COM object in the same memory space of the process itself. So there is no need for proxies or stub for communication.
- *Out of process* call: the COM object is either loaded in different memory space in the local machine, called *Local* call, or in different memory space in a remote machine, called *Remote* call.

1.2.2. Common Object Request Broker Architecture (CORBA)

The Common Object Request Broker Architecture (CORBA) [20] is open distributed object computing infrastructure being standardized by the *Object Management Group* (*OMG*). CORBA is an established standard allowing object-oriented distributed systems to communicate through the remote invocation of object method. CORBA specifies a system that provides interoperability between objects in a heterogeneous, distributed environment and in a way transparent to the programmer. Its design is based on *OMG Object Model.*

OMG Object Model defines common object semantics for specifying the externally visible characteristics of objects in a standard and implementation-independent way. In this model *clients* request services from *objects* (which will also be called *servers*) through a well-defined interface. This interface is specified in *OMG IDL* (*Interface Definition Language*). A client accesses an object by issuing a *request* to the object. The request is an event, and it carries information including an operation, the *object reference* of the service provider, and actual parameters (if any). The object reference is an object name that defines an object reliably.

The *Object Management Architecture* (*OMA*) is a high-level vision of a complete distributed environment. It consists of four components that can be roughly divided into two parts: *system oriented components* (*Object Request Brokers* (*ORB*) and *Object Services*), and *application oriented components* (*Application Objects* and *Common Facilities*). ORB is the one which constitutes the foundation of OMA and manages all communication between its components. It allows objects to interact in a heterogeneous, distributed environment, independent of the platforms on which these objects reside and techniques used to implement them. In performing its task it relies on Object Services which are responsible for general object management such as creating objects, access control, keeping track of relocated objects, *etc*. Common Facilities and Application Objects are the components closest to the end user, and in their functions they invoke services of the system components.

This paper is organized as follows. The Recovery Block (RB) technique effectiveness is discussed in Section 2. The Component-Based Recovery Block (CBRB) technique is given in Section 3, followed by the conclusion in Section 4.

2. Recovery Block (RB) Technique Effectiveness

The series of Recovery Block (RB) technique studies we conducted show that the effectiveness of the technique depends on the degree of module diversity and low probability of modules correlated faults. Hence, the probability of failing the acceptance test (AT) is independent among modules.

Furthermore, the studies allow us to identify several issues that require resolution in order to attain successful software RB fault tolerance. The following are the principle requirements for successful implementation of the technique.

1. Independence of specification writing. The most important issue of the independence design faults is a complete and accurate specification of the requirements. The independent writing of specification and subsequent comparison of the specification is expected to increase dependability of the specifications.
2. Independence of design. The efforts for design independence call for the use of diverse algorithms, programming languages, compilers, design tools, implementation techniques, test methods, *etc*. Also, it includes employing independent (noninteracting) programmers or designers, with diversity in the experience and training. Furthermore, wide geographical spread and diverse ethnic backgrounds also are desirable.

3. Low correlated faults. The effectiveness of the RB scheme depends on the validity of the conjecture that residual software faults in different modules will cause very few, if any, similar errors to fail the acceptance test. Substantial design diversity is expected when the software modules produced at widely separated locations, by programmers with different training and experience who use different programming languages.
4. Hardware faults protection. Running a primary module and alternate modules on different machines provides protection against hardware faults.
5. Modular software. Modifying existing software is a difficult task, and software existing in multiple modules is even more difficult. Modular specification will help in modifying software; only few software modules (related modules) need to be modified.
6. Cost of implementation. Generating a primary module and *N* alternate modules of an application instead of a single module (traditional software) will increase the cost of software.

3. Component-Based Recovry Block (CBRB) Technique

Being motivated by the effect of module diversity and low probability of modules correlated faults on the reliability of the Recovery Block (RB) technique and the features of the component technology, in this work we introduce a scheme called Component-Based Recovery Block (CBRB) scheme for improving system reliability.

The CBRB scheme works as follows. Every program or a program module is made up of independently (diversely) designed components performing the same task: a primary component (may be the fastest but the most complex) and some alternate components (may be slower but less complex). First, the primary component is run followed by the acceptance test (AT). If the module passes the AT, it is considered reliable; otherwise, it is considered faulty and the first alternate component is run. Similarly, if it passes the AT, it is considered reliable; otherwise, it is considered faulty and the next alternate component is run. Components are run in the order explained above. If any of the components passes the AT, the output of that component is considered reliable and used. Otherwise, an error message is reported and no output is used, see Figure 4.

3.1. Component-Based Recovery Block (CBRB) Technique Reliability Evaluation

In the Component-Based Recovery Block (CBRB) technique, different developers write specifications and construct components for every program or program module. Thus, the first requirement for the RB scheme successful implementation (independence of specification writing) is meet.

Since in the CBRB technique components for every program or program module are constructed by different developers (with diversity in experience and training, and widely geographically spread) using different algorithms, programming languages, design tools, compilers, implementation techniques, test approaches, *etc*, the second requirement for the RB scheme successful implementation (independence of design) is meet too.

Furthermore, in the CBRB technique different developers construct components for every program or program module with diversity in experience and training, widely geographically spread, and using different programming languages. Thus, the third requirement for the RB scheme successful implementation (low correlated faults) will be meet.

Also, in the CBRB technique different components (modules) could be located on different (remote) machines. These components could be loaded into the local machine or executed in remote sites. Thus, the forth requirement for the RB scheme successful implementation (hardware faults protection) will be meet.

Due to that in the CBRB scheme software consists of components, any future changes in the specification will accompany replacing few of the components (related to the changes) with new components (well tested) instead of modifying the existing modules. Replacing the existing components with new ones will speed up the process of modification. In other words, the system availability will be improved. Thus, the fifth requirement for the RB scheme successful implementation (modular software) is meet too.

Finally, in the CBRB technique, instead of developing *N+1* different modules of each program or program module, we can use (buy or rent) already exist components (modules). Which will decrease the implementation cost. Thus, the sixth requirement for the RB scheme successful implementation (cost of implementation) will be meet.

From the above we can see that the CBRB technique meets the above principle requirements for RB scheme successful implementation. Thus, improving system reliability through the CBRB technique is expected.

4. Conclusion

Motivated by the effect of module diversity and low probability of modules correlated faults on the reliability of the Recovery Block (RB) technique and the features of component-based software, we introduced a more reliable scheme called Component-Based Recovery Block (CBRB) scheme. The proposed technique is based on the component software technology and the RB scheme. We showed that the CBRB approach satisfies all the principle requirements for RB scheme successful implementation. Thus, under the proposed scheme systems reliability improvement is expected.

Furthermore, due to the need for increasing reliability of systems running critical applications, the emergence of component technology is expected to be boosted.

This work will serve as the foundation for future research work including:

- Experimental studies needed to be conducted to study the effect of using the CBRB scheme on system reliability and performance.
- Modeling system reliability and performance of the proposed scheme.

5. References

[1] J. Voas. Software Fault Tolerance: Making Software Behave. IEEE Software, Volume (18): 18-19, 2001.

[2] A. Avižienis. The Methodology of N-Version Programming, chapter in: M. R. Lyu, (Ed.) Software Fault Tolerance. John Wiley & Sons, 1995.

[3] B. Randel and J. Xu. The Evaluation of the Recovery Block Concept, chapter in M. R. Lyu, (Ed.) Software Fault Tolerance, John Wiley & Sons, 1995.

[4] S. Brilliant, J. C. Knight, N. G. Leveson. Analysis of Faults in N-Version Programming Software Experiment. IEEE Transactions on Software Engineering, 1990.

[5] N. G. Leveson. Software Fault Tolerance: The Case for Forward Recovery. In proceedings of the AIAA Conference on Computers in Aerospace, 1983.

[6] L. Sha. Using Simplicity to Control Complexity. IEEE Software, Volume (18): 20-28, 2001.

[7] P. A. Lee and T. Anderson. Fault Tolerance – Principles and Practices. Springer-Verlag/Wien, 1990.

[8] O. A. Abulnaja, S. H. Hosseini, K. Vairavan. Scheduling Algorithms for Recovery Block Scheme. In proceedings of the IASTED International Conference on Modelling and Simulation, 1996.

[9] J. J. Horning, H. C. Lauer, P. M. Melliar-Smith, B. Randell. A Program Structure for Error Detection And Recovery. In Lecture Notes in Computer Science, Volume (16): 171-187, Springer-Verlag, 1974.

[10] B. Randel. System Structure for Software Fault Tolerance. IEEE Transactions on Software Engineering, Volume (SE-1): 220-232, 1975.

[11] T. Anderson, D. N. Halliwell, P. A. Barrett, M. R. Moulding. An Evaluation of Software Fault Tolerance in a Practical System. In proceedings of the 15th International Symposium on Fault Tolerant Computing, 1985.

[12] R. H. Campbel, K. H. Horton, G. G. Belford. Simulations of a Fault-Tolerant Deadline Mechanism In proceedings of the 9th International Symposium on Fault Tolerant Computing, 1979.

[13] H. Hecht. Fault-Tolerant Software. IEEE Transactions on Reliability, Volume (R-28): 227-232, 1979.

[14] H. Welch. Distributed Recovery Block Performance in a Real-Time control Loop. In proceedings of the Real-Time Systems Symposium, 1983.

[15] R. K. Scott, J. W. Gault, D. F. McAllister. Fault Tolerant Software Reliability Modeling. IEEE Transactions on Software Engineering, Volume (SE-13): 582-592, 1987.

[16] J. Hudak, B. H. Suh, D. Siewiorek, Z. Segall. Evaluation & Comparison of Fault-Tolerant Software Techniques. IEEE Transactions on Reliability, Volume (42):190-204, 1993.

[17] G. Pucci. A New Approach to the Modeling of Recovery Block Structure. IEEE Transactions on Reliability, Volume (18): 159-167, 1992.

[18] S. Islam, and H. Ammar. Performability of Integrated Software-Hardware Components of Real-Time Parallel and Distributed Systems. IEEE Transactions on Reliability, Volume (41): 352-362, 1992.

[19] W. Rubin and M. Brain, Understanding DCOM, Prentice Hall PTR, 1999.

[20] R. Geraghty, S. Joyce, T. Moriarty, G. Noone, COM-CORBA Interoperability. Prentice Hall PTR, 1999.

```
ensure <acceptance test> by
   <primary module>
   else by
   <alternate module 1>
   .
   .
   .
   else by
   <alternate module N>
   else error
```

Figure 1. RB Technique

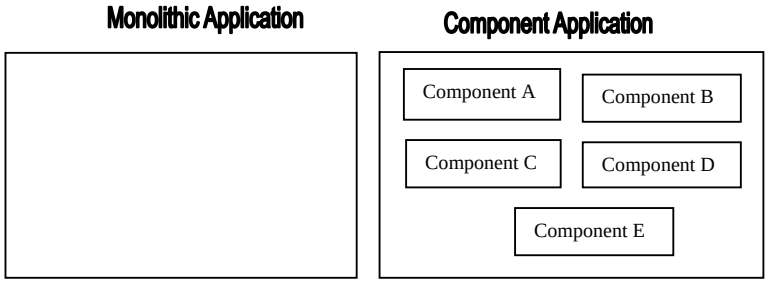

Figure 2. Breaking Monolithic Application into Components

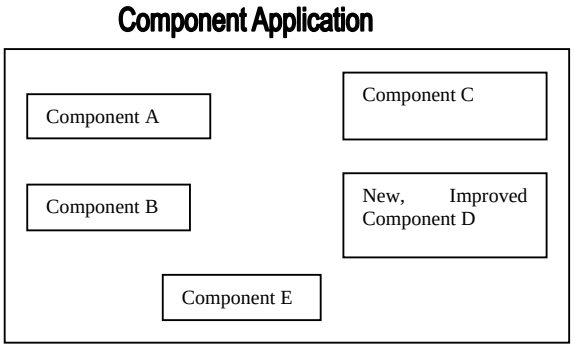

Figure 3. New, Improved Component D Replaced Old Component D

```
ensure <acceptance test> by
   <primary component>
   else by
   <alternate component 1>
   .
   .
   .
   else by
   <alternate component N>
   else error
```

Figure 4: CBRB Technique