

Middleware of Real-Time Object Based Fault-Tolerant Distributed Computing Systems: Issues and Some Approaches

K. H. (Kane) Kim

DREAM Lab, UCI
Irvine, CA, USA
<http://dream.eng.uci.edu>

(Keynote)

ABSTRACT : At this turn of the century the object-oriented (OO) distributed real-time (RT) programming movement is growing rapidly along with the networked embedded systems market. The motivations are reviewed and then a brief overview is given of the particular programming scheme which this author and his collaborators have been establishing. The scheme is called the time-triggered message triggered object (TMO) programming scheme and it is used to make specific illustrations of the issues and potentials of OO RT programming.

Fault tolerance capabilities are required in many distributed RT computing applications. At this time the development of middleware which is capable of supporting reliable fault-tolerant execution of application-level RT distributed objects is an important challenge to the research community. Some major issues that need to be resolved and some promising approaches are discussed.

Keywords: abort, availability, detection, distributed computing, fault, network surveillance, object, real time, recovery, reliability, replication, transaction, tolerance.

1. Introduction

For much of the last quarter of the 20th century, the fault tolerance related concerns of main-stream computing industry were to merely maintain the integrity of the DBMS. It sufficed for them to facilitate *clean abort* of transactions whenever faults in intermediate computation results or uncommitted data were found. They did not feel that additional execution overhead and hardware and software costs involved in facilitating *higher-coverage fault-tolerant (FT) computing*, e.g., automatic retry of a failed transaction, were worthwhile. If they tried to do that, their products would not be competitive in the market of those times.

At this entry point of the 21st century, the situation is different. There is still no sign of growth (or one can even say, resurrection) of specialized FT computing industry. However, the interests of main-stream computing industry in FT distributed computing (DC) are now growing fast for at least two reasons:

(1) Rapid growth of the *Web server* market and customers' growing demands for *high-availability Web servers*, and

(2) Rapid growth of RT computing applications that started around mid-1990's, especially growing demands for computer-embedded communication-equipped devices / systems in this new decade.

The main-stream computing industry is trying to meet these demands by incorporating cost-effective FT DC mechanisms within the framework of general-purpose hardware facilities and general-purpose operating system (OS) architecture [Kim00d].

Some say that end users lose patience when Web sites handling competitive commercial activities take longer than 8 seconds to show results. This means that

(1) Such Web sites must be up "all the time", i.e., meet high-availability requirements; and

(2) Web sites must respond fast even when they are accessed by a large number of clients concurrently.

Therefore, the main-stream computing industry is now becoming better motivated to explore higher-coverage FT DC approaches than ever before.

In this new decade, the real-time (RT) computing application market is no longer a negligible market even for major platform vendors. In RT applications, mere clean abort of transactions is rarely an acceptable approach because an abortion of a transaction leads more often than not to abandoning the application. Most sizable RT computing applications are now of DC type. Again, *higher-coverage FT DC* is becoming a lively field.

The field of computer-embedded communication-equipped system engineering has been growing particularly fast in recent years. As a result, industry has felt an acute need for RT distributed programming and software engineering methods which are at least *multiple times more effective* than currently widely practiced programming techniques. This new-generation RT distributed software engineering method must be based on a "general high-level programming style" which can be accommodated with minimal efforts by current-generation business application programmers (using C++ and Java) rather than on a style that has been practiced by assembly language or low-level language programmers.

The fact that *distributed objects* represent a higher-level structure for distributed programs than *distributed processes* do have been widely recognized by the industry in the past 10 years, e.g., technology movements such as

CORBA [OMG01, Nar99, Sol95], COM [Ses97], SOAP, dot-NET (by Microsoft), and RMI by Sun Microsystems. Naturally, researchers started searching for extensions of distributed objects that allow unambiguous specification of timing requirements imposed on various computations units [IEE00, ISO, Kim00d, WOR].

Nearly all RT fault tolerance techniques practiced in Century 20 were coupled with the approaches for process structuring of RT DC software. Therefore, adapting the existing RT fault tolerance techniques for integration into the powerful RT OO DC structure is an important challenge in this new decade. The purpose of this paper is to take an overview of the technical issues collectively forming this challenge.

Main issues in realizing RT fault tolerance in RT OO DC systems, especially facilitating such FT computing by new middleware capabilities, are discussed in Section 2. Whenever the discussion on the issues sounds ambiguous, readers are advised to look at the particular RT OO distributed programming model which this author and his collaborators have been establishing in recent years. The scheme is called the *time-triggered message triggered object* (TMO) programming scheme and is discussed in [Kim97b, Kim00b]. It is a general-style RT DC extension of the conventional OO programming approach. It is a syntactically simple and natural but semantically powerful extension of the conventional object programming approaches. The TMO offers a powerful structure which is capable of representing all conceivable practical RT and non-RT applications in easy-to-analyze forms. In fact, the scheme facilitates uniform structuring of

- both RT and non-RT distributed application systems,
- both control computer systems and their application environment simulators, and
- requirement specifications and system designs arising at various phases of system engineering.

TMOs are high-level program constructs in that conventional low-level program abstractions such as processes, threads, priorities, and socket communication protocols are transparent to TMO programmers.

A particular set of middleware approaches for supporting RT fault tolerance that are being explored in the author's laboratory are briefly sketched in Section 3. More detailed discussions on some challenging issues are also given. The paper is concluded in Section 4.

2. Issues in realization of fault tolerance in RT OO DC systems

2.1 Clean abort, high-availability server, and RT recovery

As already mentioned, high-availability Web servers must be capable of doing more than clean abort. If a node crashes, the impacted on-going transactions must be cleanly aborted and the server function of the crashed node must be resurrected in another node within a reasonable amount of time. Therefore, this high-

availability server approach is aimed for *higher coverage* in FT DC than the coverage goal of the conventional server relying on clean abort only. The former leads clients to experiencing disconnection from the server for shorter duration and incurs less costs to the applications / missions than the latter does.

When wireless network components are used in Web server applications, the fault rate and the needs for fault tolerance mechanisms tend to become more significant.

RT computing applications, e.g., video-conferencing, voice over IP (internet protocol), factory automation, defense applications, etc., require even higher coverage in FT DC. Attempts for automated retry of a failed transaction or concurrent redundant tries of a transaction are usually essential. That is, attempts to avoid any loss of a transaction are desirable. Therefore, forward or backward *RT recovery* from faults is a usual attempt.

This means that the interests of main-stream computing industry in FT DC technologies have been advancing in the past two decades as follows [Kim00d]:

Clean abort technologies

- high-availability server technologies
- RT recovery technologies.

Of course, research communities dealt with all three types of FT DC approaches and various proven or promising solutions have been produced. However, the amount of research invested in the three areas has been largely proportional to the degree of interests of the main-stream industry shown in the areas.

In RT FT DC systems, fault detection and recovery actions should ideally be executed such that intended output actions of RT computations always take place on time in spite of fault occurrences [Kim94]. When it is not feasible, the fault tolerance actions which lead to the least damages to the application missions / users must be attempted.

In order to be practically useful in RT DC systems, a fault detection technique must at least yield a tightly bounded *detection latency* and a recovery technique must at least yield a tightly bounded *recovery time* [Kim99a, Kim00a]. Only a tiny fraction of the research conducted in Century 20 dealt with rigorous analyses of detection latency bounds and recovery time bounds.

Even in non-RT application systems such as many Web server applications, detection latency and recovery time are becoming important performance metrics since they are major contributors to the server-down time. However, in such environments, average server-down times are more relevant than tight bounds on the server-down time are.

In fact, there are signs that even the major vendors of OS and communication infrastructure are gradually stepping up their efforts in making the timing behavior of their products more predictable. The justifications for keeping their products to continue to show wildly

unpredictable timing behavior are losing strength. This will make the boundary between FT DC approaches used in non-RT application systems and those used in RT application systems to become blurred somewhat.

2.2 Types of fault symptoms to deal with

In designing RT OO DC systems such as those consisting of TMO networks running on networks of middleware-kernel-hardware platforms, major types of fault symptoms to consider are the six types discussed below. It should be noted that in terms of the technical difficulties of realizing RT fault tolerance, the middleware-level approaches and the application-software-level approaches are harder than the hardware approaches and the kernel-level approaches, as depicted in Figure 1. However, various cost factors put middleware approaches in highly favorable positions. That is why the middleware support for RT OO fault tolerant DC is becoming a very popular area of research in the early stage of the 21st century.

(1) Crash faults of hardware components

Here typical hardware components are processors and communication links which may be those between computing nodes or those connecting computing nodes to a network bus. These faults are among the most likely faults to be encountered in operating RT DC systems. Detecting the symptoms of these faults should be an important responsibility of middleware or OS kernels.

Past research has produced various techniques for detection of these fault symptoms. Some are based on centralized analysis [Hec91] and others are based on partially or fully decentralized analysis. The latter type of techniques are often called *network surveillance* [Kim94, Kim00d] or *membership maintenance* techniques [Kop97, You00]. To be more specific, network surveillance is basically a (partially or fully) decentralized mode of detecting faulty and repaired status of distributed computing components. It is aimed for minimizing the periods during which faulty components (e.g., processing nodes and communication links) are lurking in distributed computing systems. This means to facilitate fast *learning* by each interested fault-free node of the *faults* or *repair completion* events occurring in other parts of the distributed computing system.

The important metrics in this area is again the *detection latency bound* [Kim94, Kim99a, Kim00d]. There are some techniques which yield to rigorous quantitative analyses of fault coverage [Kim94, Kim99a, Kop93, Kop97]. However, this area requires further research, especially in cost-effective integration of the techniques mentioned with other fault detection and recovery techniques.

(2) Crash faults of OS (including middleware) and application server objects

Difficulty of Achieving Real-Time FT

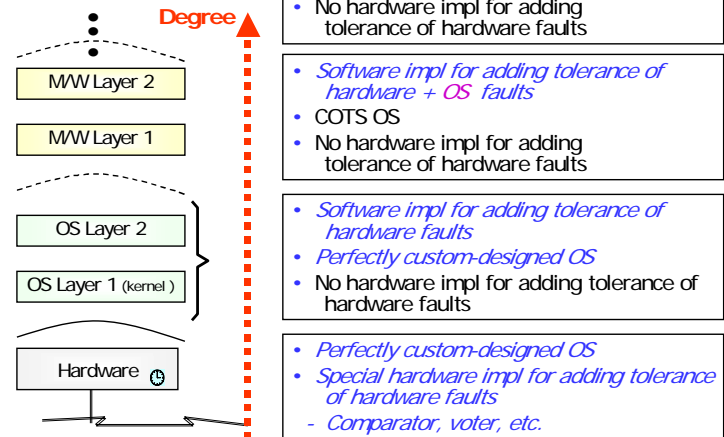


Figure 1. The relative difficulties of RT fault tolerance approaches at different levels

These crash faults may occur for two reasons: (a) Crash faults of hardware components and (b) design faults lurking in OS or application server objects.

The detection of and recovery from the complete crash of an OS can be done in the same manner as in the case of (1). If a partial crash of an OS, i.e., the crash of one or a few subsystems of an OS, is detected by an external node facility or the surviving facility on the same node, cooperating nodes must "convert" it into a full crash of the node.

In the case of a crash of an application server object, then the *DC configuration manager* (DCCM) must resurrect the object in another healthier node. Although no conceptual problems are present here, experiences of actual implementation are scarce. In fact, how to minimize disruptions or losses in the services to the clients during this resurrection has been a poorly studied subject.

If design faults were the cause of the crash of an application server object, then resurrection on another node will not help the situation. This of course is a long-term research issue.

(3) Transient deafness of hardware components

Transient deafness of hardware components are manifested when processors or communication links do not respond to service requests for some limited periods. Message losses could also be the resulting symptoms. These faults are often masked by retry capabilities of middleware or OS kernels. There is also some advocacy for making double submission of service requests all the time regardless of whether a deaf symptom is observed or not [Kop97]. Here the argument is that the worst-case

performance of the system does not get degraded and it can be analyzed more easily than in the case of doing dynamic need-based retries.

(4) Transient deafness of OS (including middleware) and application server objects

First, transient faults of some hardware components may lead to these symptoms. Secondly, design faults could lead to these but with reasonable-quality products, only minor symptoms of this kind may be exhibited. This author has not seen any report of a success case where these faults were masked by retry capabilities of middleware or OS kernels. It is considered a prudent approach to design a system such that the parties which detect these fault symptoms treat the relevant OS or application server objects as crashed items.

(5) Action timing faults of application server objects

These faults occur when the service results from the RT server objects do not return to the RT clients in time. Given the quality of commercially available platforms including OS, node hardware, and communication infrastructure as well as RT software engineering tools at this time, these faults are expected to be encountered *more frequently* than crash faults discussed above. Therefore, the designer of RT client programs must anticipate these faults and make provision for continued computation in spite of these fault occurrences. The designer may make the client program to try to get a service from an alternative source when such a fault occurs. The designer must of course understand the time budget available for this second attempt.

(6) Value faults of application server objects

These faults occur when the logical values of the service results from the RT server objects are incorrect. They occur typically due to design faults lurking in the server objects. Handling such faults is in the domain of *software fault tolerance* [Avi88, Kim95, Ran75, Ran95], a subject which is inherently very difficult and complex to the point of appearing nebulous sometimes. An effective method for validating the schemes aimed at handling design faults is under continuous search. Some successful demonstrations of the capabilities for detecting symptoms of design faults at run time have taken place but they did not include any successful RT recovery actions other than stopping the system operation. Nevertheless, this area deserves attacks by multiple determined resourceful research groups.

2.3 Integration of component technologies: A challenge

Handling some of the faults discussed in the preceding section in cost-effective manners has remained a technical challenge. A more important challenge in development of the RT FT DC technology appears to be

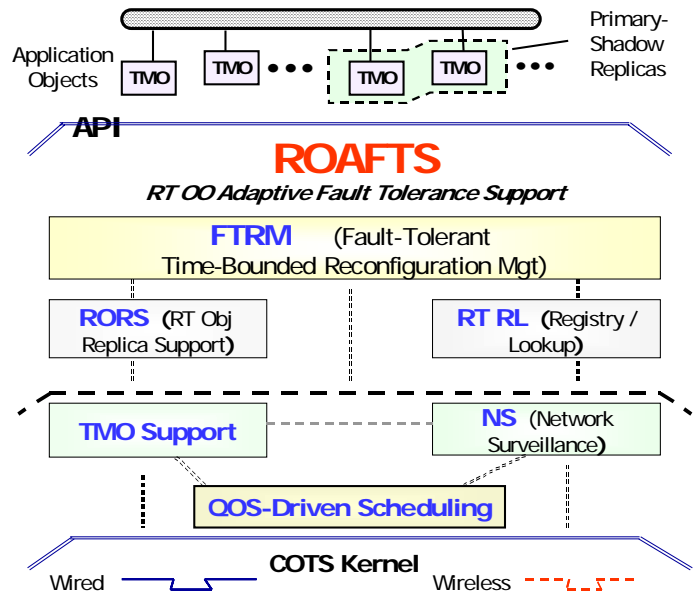


Figure 2. A high-level view of ROAFTS Middleware

the *integration*. The state of the art has reached the point where promising component techniques which are effective in achieving specialized fault tolerance capabilities of subsystems when used in isolation, are available but their cost-effective integration remains insufficiently done. There is of course a large room for research to mature and further enhance these component techniques. However, a more urgent and important issue from the viewpoint of balanced technology development is cost-effective integration of such component techniques.

3. ROAFTS approach and additional open research issues

The approach being explored by the author and his collaborators to tackle the challenges discussed in the preceding section are now briefly sketched in this section. Several challenging issues open for future research are also mentioned.

3.1 ROAFTS architecture and the layering issue

The architecture and a prototype implementation of a middleware which supports reliable fault-tolerant execution of TMO-structured distributed applications, have been developed by this author and his collaborators. This middleware has been named ROAFTS (*Real-time Object-oriented Adaptive Fault Tolerance Support*) [Kim00d] and it is still evolving at a rapid rate.

- (i) extensive system monitoring, including cooperative distributed detection of resource failures and transient overloads; and
- (ii) adaptive configuration management including allocation of available resources to the active applications

such that the minimum required quality of services (QoS's) of critical components may be maintained for longest possible periods. The main components of ROAFTS are depicted in Figure 2.

The *network surveillance* component uses the *supervisor-based network surveillance* (SNS) scheme [Kim99a, Kim00a]. This scheme provides the capability for detecting crash faults of hardware components and OS. It yields a tightly bounded detection latency bound. The SNS scheme is a semi-centralized RT NS scheme effective in a variety of point-to-point networks and can also be adapted to broadcast networks. This scheme is highly scalable and can be implemented entirely in software using COTS components without requiring any special-purpose hardware support. Prototype implementations in the author's lab have been running reliably for several years.

Active replicas of an application TMO are easily created and supported by the *primary-shadow TMO replication* (PSTR) support component [Kim97a, Kim00a]. The PSTR scheme provides the capability for detecting all the faults which are not detected by SNS and also RT recovery from them. It yields a tightly bounded recovery time bound. The primary-shadow replication principle was earlier established in the *distributed recovery block* (DRB) scheme [Kim94, Kim95] (an extension of the *primary-shadow pair of self-checking processing-units* (PSP) scheme) which was effective in process-structured RT DC systems. One prototype implementation of the PSTR support component which supports primary-shadow replication of a simple type of TMOs has been running for three years. This prototype supports *on-line joining / marriage of a new shadow TMO* to the existing TMO. The second prototype which supports a fully general TMO has been running for half a year. The on-line shadow-joining capability in this prototype is still evolving.

The reconfiguration manager that complements the network surveillance middleware component and the PSTR support component to enable full fault-tolerant operations is the *fault-tolerant time-bounded reconfiguration management* (FTRM) middleware component. The FTRM is responsible for fault containment and establishment of an operating platform network and object network configuration. It will react to the detection of various types of faults by the network surveillance component and the PSTR support component. The prototype implementation developed is of primitive nature and will evolve for a while.

The layering of components in the ROAFTS architecture was influenced by our desire to keep the temporal behavior analysis to be relatively easy. The research community is expected to explore many different approaches for modularizing and layering components. It

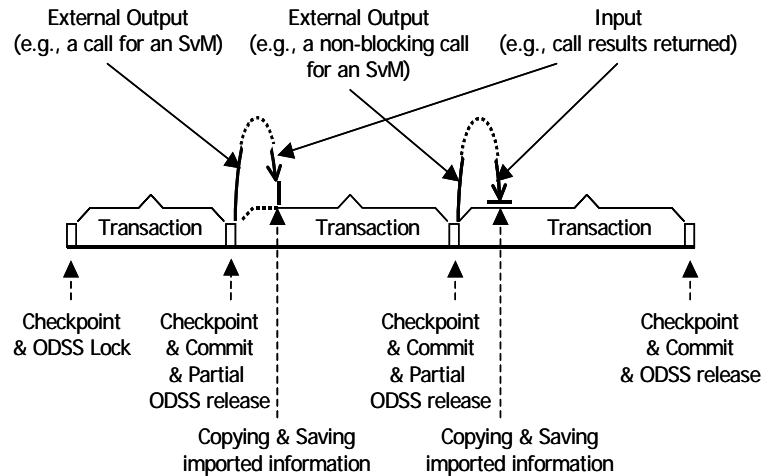


Figure 3. The multi-transaction model

is hoped that the community also establish quantitative approaches for evaluating trade-offs existing among different middleware architectures.

3.2 Detection of faults in TMO and challenging issues in recovery

The primary - shadow cooperative relationship exists at the object-method level. Under the PSTR scheme, when the primary method execution runs into a fault, the shadow method execution which has been progressing in a healthy mode takes over the role of the primary. This role switch in one method should lead to the role switch in all the rest of the TMO pair. However, in general multiple methods in the same TMO may be in concurrent execution and thus instant role-switch in all other methods is not feasible. Research is under way to understand the implications of this practical constraint.

The PSTR also dictates execution of an object-method in the form of a chain of transactions as depicted in Figure 3. When the primary TMO and the shadow TMO execute two different versions of application code for the sake of avoiding common design faults, this implies certain restrictions on the possible diversity in the algorithm / code structures used in the two versions. Research is continuing to understand these implications.

4. Conclusion

In this paper some major issues in realization of fault tolerance in RT OO DC systems have been reviewed. Some techniques which can serve as cornerstones in building up a desirable technology have also been reviewed. While many component techniques have emerged, their cost-effective integration and subsequent optimization remain huge challenges. Much more research is needed and strong demands are coming from potential market areas.

Acknowledgements: The research work reported here was supported in part by the NSF under Grant Numbers

99-75053 (NGS) and 00-86147 (ITR), and in part by the US DARPA under Contract F33615-01-C-1902 monitored by AFRL.

References

[Avi88] Avizienis, A., Lyu, M.R., and Schutz, W., "In Search of Effective Diversity: A Six-Language Study of Fault-Tolerant Flight Control Software.", *Proc. IEEE CS 18th Int'l Symp. on Fault-Tolerant Computing (FTCS-18)*, pp.15-22.

[Hec91] Hecht, M., et al., "A Distributed Fault Tolerant Architecture for Nuclear Reactor and Other Critical Process Control Applications.", *Proc. IEEE CS 21st Int'l Symp. on Fault-Tolerant Computing (FTCS-21)*, June 1991, Montreal, pp.462-469.

[IEE00] 'A special issue of Computer (a magazine of IEEE Computer Society) on Object-oriented Real-time distributed Computing', June 2000.

[ISO] ISORC '98 (*IEEE CS Int'l Symp. on Object-oriented Real-time distributed Computing Series*); 1st held in April 1998, Kyoto, Japan; 2nd in May 1999, St. Malo, France; 3rd in March 2000, Newport Beach, CA, and 4th in May 2001, Magdeburg, Germany. Proceedings are available from IEEE CS Press.

[Kim94] Kim, K.H., "Action-Level Fault Tolerance", Ch. 17 in Sang H. Son, 'Advances in Real-Time Systems', Prentice Hall, 1994, pp.415-434.

[Kim95] Kim, K.H., "The Distributed Recovery Block Scheme", Ch. 8 in Michael R. Lyu, ed., 'Software Fault Tolerance', 1995, pp. 189-209.

[Kim97a] Kim, K.H., and Subbaraman, C., "Fault-Tolerant Real-Time Objects", *Communications of the ACM*, January 1997, pp. 75-82.

[Kim97b] Kim, K.H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No.8, August 1997, pp. 62-70.

[Kim99a] Kim, K.H. and Subbaraman, C., "Dynamic Configuration Management in Reliable Distributed Real-Time Information Systems", *IEEE Trans. on Knowledge and Data Engr.*, Vol.11, Jan./Feb. 1999, pp.239-254.

[Kim99b] Kim, K.H. Ishida, Masaki, Liu, Juqiang, "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", *Proc. 2nd IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing (ISORC '99)*, St. Malo, France, May, 1999, pp.54-63.

[Kim00a] Kim, K.H. and Subbaraman, C., "The PSTR/SNS Scheme for Real-Time Fault Tolerance via Active Object Replication and Network Surveillance", *IEEE Trans. on Knowledge and Data Engr.*, Vol.12, No.2, Mar./April 2000, pp.145-159.

[Kim00b] Kim, K.H., "APIs for Real-Time Distributed

Object Programming", *IEEE Computer*, June 2000, pp.72-80.

[Kim00c] Kim, K.H., "Object-Oriented Real-Time Distributed Programming and Support Middleware", *Proc. ICPADS 2000 (7th Int'l Conf. on Parallel & Distributed Systems)*, Iwate, Japan, July 2000, pub. by IEEE CS Press (keynote paper), pp.10-20.

[Kim00d] Kim, K.H., "Issues Insufficiently Resolved in Century 20 in the Fault-Tolerant Distributed Computing Field", *Proc. SRDS 2000 (19th IEEE CS Symp. on Reliable Distributed Systems)*, Nuremberg, Germany, Oct. 2000, pp.106-115 (Invited paper).

Presentation slides are available from

http://dream.eng.uci.edu/TMO/pdf/srds2000_slides.pdf .

[Kop93] Kopetz, H., and Gruensteidl, G., "TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems", *Proc. 23rd IEEE Int'l Symp. on Fault-Tolerant Computing (FTCS-23)*, Toulouse, France, 1993, pp. 524-533. A revised version appeared in *IEEE Computer*, vol. 27 (1), 1994, pp. 14-23.

[Kop97] Kopetz, H., 'Real-Time Systems', Kluwer Academic Pub., 1997.

[Nar99] Narasimhan, P., Moser, L. E., and Melliar-Smith, P. M., "Using Interceptors to Enhance CORBA," *IEEE Computer*, July 1999, pp. 62-68.

[OMG01] Object Management Group, 'The common Object Request Broker: Architecture and Specification', Revision 2.5, Sep., 2001, available from www.omg.org.

[Ran75] Randell, B., "System Structure for Software Fault Tolerance.", *IEEE Trans. on Software Engineering*, June 1975, pp.220-232.

[Ran95] Randell, B. and Xu, J., "The Evolution of the Recovery Block Concept", Ch. 1 in M. R. Lyu ed., 'Software Fault Tolerance', John Wiley & Sons, Chichester, England, 1995, pp.1-21.

[Ses97] Sessions, R., 'COM & DCOM; Microsoft's Vision for Distributed Objects', John Wiley & Sons, Inc., New York, Oct. 1997.

[Sol95] Soley, R. ed., 'Object Management Architecture Guide', 3rd edition, John Wiley & Sons, New York, 1995.

[WOR] WORDS (*IEEE CS's Workshop on Object-oriented Real-time Dependable Systems Series*); 1st held in Oct. '94, Dana Point; 2nd in Feb. 1996, Laguna Beach; 3rd in Feb. 1997, Newport Beach; 4th in Jan. 1999, Santa Barbara; 5th in Nov. 1999, Monterey; 6th in Jan. 2001, Rome, Italy. Proceedings are available from IEEE CS Press.

[You00] H. Kim, D. Lee, and H.Y. Youn, and D. Lee, "A Scalable Membership Service for Group Communications in WANs," PRDC 2000 (Pacific Rim Int'l Symp. on Dependable Computing, Dec. 2000), pp. 59-66.