# Estimating Fault-Detection and Fail-Over Times for Nested Real-Time CORBA Applications

Sukanya Ratanotayanon (speaker)
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213–3890
*Tel:* +1-412-268-8801 *Email:* sratanot@andrew.cmu.edu

Priya Narasimhan
Electrical & Computer Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15213–3890
*Tel:* +1-412-268-8801 *Email:* priya@cs.cmu.edu

***Abstract*— Today's middleware applications tend to be complicated, and consist of tiers that form a nested chain of objects or processes. For a real-time nested application, predictability is crucial, even when faults and restarts occur. Nested applications make it challenging to predict the processing time when faults occurs. The fault-detection and recovery times are influenced by the number of tiers, the tier in which the fault occurs, the state of the end-to-end processing when the fault occurs, and the processing time of each tier. We investigate the behavior of nested CORBA applications when faults occur in different tiers, and at different stages of the nested processing. We present a model for predicting the worst-case fault-detection and fail-over times for such nested applications.**

**Keywords:** real-time, fault-tolerance, CORBA, nested applications, worst-case performance

## I. Introduction

Today's distributed mission-critical applications are very complex, and are increasingly built as nested applications with multi-tier architectures to get better performance, maintainability and scalability. The integration of individual subsystems to perform autonomous tasks is another reason for nesting in such applications. To simplify developing distributed applications, middlewares are widely adopted by application developers. Commercial off-the-shelf (COTS) middleware, such as the Common Object Request Broker Architecture (CORBA) [3], renders transparent the low-level details of networking, distribution, physical location, hardware, operating systems, and byte order.

Many applications, such as radars, telephone switching and stock trading, require both real-time and fault tolerance. To obtain real-time behavior, predictability must be maintained, even when faults occur in the system. This is difficult enough for non-nested applications because of the conflicts imposed by real-time and fault-tolerant properties [2]. However, this becomes far more challenging for nested applications because more factors affect their predictability when faults occur. These factors include the number of nested tiers, the precise tier in which the fault occurs, and the stage of end-to-end processing when the fault occurs.

We investigate the behavior of nested real-time CORBA applications in the presence of faults. This paper focuses on estimating the impact of a fault when the fault occurs in different tiers, and at different stages of the end-to-end nested processing. One of our contributions is a model to predict the worst-case fault-detection and fail-over times, in terms of the number of tiers and each tier's processing time; our experiments have served to validate this model.

## II. Background

CORBA applications consist of objects distributed across the system, with client objects invoking server objects that return responses to the client objects after performing the requested operations. The Object Request Broker (ORB) acts as an intermediary in the communication between a client and a server, transcending differences in their programming language (language transparency) and their physical locations (location transparency). The Portable Object Adapter (POA), a server-side entity that deals with the actual implementations of a CORBA server object, allows application programmers to build implementations that are portable across different vendors' ORBs. CORBA's General Internet Inter-ORB Protocol (GIOP) and its TCP/IP-based mapping, the Internet Inter-ORB Protocol (IIOP), allow client and server objects to communicate regardless of differences in their operating systems, byte orders, hardware architectures, etc. In order to satisfy the demand for middleware that supports the application requiring high reliability and quality of service, CORBA provides FT-CORBA [4] and RT-CORBA [5] extensions to support those applications.

The FT-CORBA specification provides reliability through the replication of CORBA objects, and the subsequent distribution of the replicas of every object across the processors in the system. The specification specifies components to support fault detection, fault notification, and replication management.The Fault Detector is capable of detecting host, process and object faults and communicates the occurrence of faults to the Fault Notifier. The Fault Detectors can be structured hierarchically, with the global replicated Fault Detector triggering the operation of local fault detectors on each processor. Any faults detected by the local fault detectors are reported to the global replicated Fault Notifier. On receiving reports of faults from the Fault Detector, the Fault Notifier filters them to eliminate any inappropriate or duplicate reports, and then distributes fault-event notifications to interested parties. The Replication Manager, being a subscriber of the Fault Notifier,

receives reports of faults that occur in the system, and can, therefore, initiate appropriate recovery actions. The Logging and Recovery Mechanisms are located underneath the ORB, in the form of non-CORBA entities, on each processor that hosts replicas. They are intended to capture checkpoints of the application, and to store them for the correct restoration of a new replica.

The Real-Time CORBA (RT-CORBA) specification aims to facilitate the end-to-end predictability of activities in the system, and to allow CORBA developers to manage resources and to schedule tasks. The specification includes a number of components, each of which must be designed or implemented by the RT-CORBA vendor to be predictable. The components include the real-time ORB (RT-ORB), the real-time POA (RT-POA), the mapping of the ORB-level priorities to the operating system's native priorities, and the server-side thread pool. In addition to the core CORBA infrastructural enhancements, the specification also includes a Real-Time CORBA Scheduling Service for the offline scheduling of the application's tasks, typically in accordance with the proven Rate Monotonic Analysis algorithm. The specification also specify two model to specify a priority that a server processes a client's invocation based on. In the client-propagated priority model, the client specifies the priority for the invocation, and the server honors this priority. In the server-declared priority model, the server specifies the priority at which it will execute the invocation. To improve the predictability of the system, clients are allowed to set timeouts to bound the amount of time that they wait for a server's response.

## III. Nested Applications

CORBA applications that are not very complicated can be built in a non-nested form to contain objects that act only either as a pure client or as a pure server. An example is a two-tier application which is separated into two parts: a pure client part (e.g., a user interface) and a pure server part (e.g., back-end processing). Such non-nested "flat" architectures are not sufficiently scalable or flexible to support complicated applications spanning several servers and with multiple clients.

To overcome the limitations of "flat" architectures, or to support the communication/sharing of data across sub-systems, more tiers are chained together to build a *nested application* that achieves a specific end-to-end function. The additional tiers provide facilities for integrating existing components, along with better performance through concurrency, maintainability through decoupled functionality, and scalability. In this nested chain, a server may process an incoming invocation from its client (preceding object) by sending an outgoing invocation to another server (following object). Thus, an object in an intermediate tier acts as both a client and a server, depending on the semantics (incoming or outgoing) of the operation. An example of an application of a pure client with $n$ nested tiers is shown in Figure 1; here, $S_i$ is a server for $S_{i-1}$ and a client for $S_{i+1}$, and $S_n$ is a pure server; note that the pure client count as tier $0$ in this system.

The term *nested operation* refers to the end-to-end processing, starting the pure client to the last server, $S_n$. Because a nested operation propagates through $n$ tiers (i.e., $n$ server objects), it can be at any one of the $n+1$ objects (including the pure client) within the chain. The term *stage* is used to refer to the precise object/tier that has finished processing its share of the nested operation, and is now ready to invoke the next tier (stage $0$ represents the start of the nested operation by the pure client). For example, for an application with a client and three nested tiers, if the invocation has been processed at the second tier, which is now ready to invoke the third tier, the nested operation is at the second stage. Once the third/last tier has processed, but not yet responded to, the invocation, the nested operation is at the third stage. Once the third tier returns the response, the notion of stage is meaningless. The remainder of the paper deals with the real-time and fault tolerance issues for nested CORBA applications at various stages and tiers of processing.

### A. Architecture for RT-FT Nested Applications

In order to satisfy high reliability and real-time requirements of mission-critical applications, we consider a combination of FT-CORBA and RT-CORBA as a starting point, as shown in Figure 2. In this architecture, RT-CORBA features, such as RT-ORB, RT-POA, priority mapping, thread pool, etc. are used to provide predictability in the fault-free case. FT-CORBA components are exploited to enable the system to tolerate faults.

To explore the fundamental issues, we focus on a simple system with stateless warm passively replicated servers. Each server has a designated primary replica that processes requests; the backup replicas remain dormant until one of them is elected as the new primary if the old primary fails. Fault detection, fault notification, and replication management con-

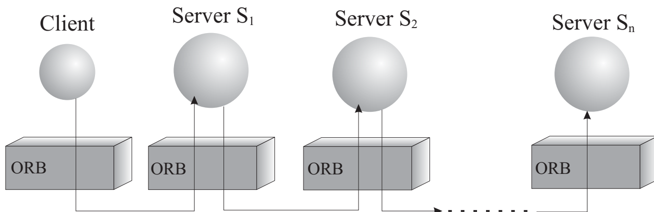


Fig. 1. A typical nested CORBA application.

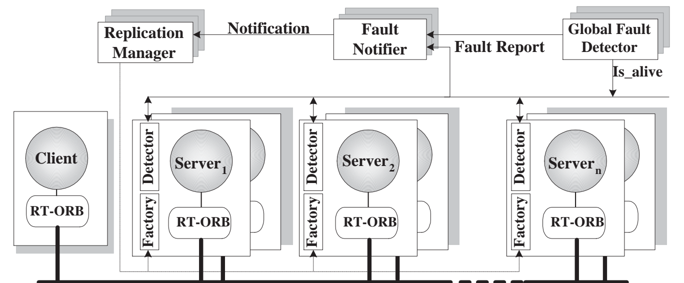


Fig. 2. Architecture for RT-FT nested CORBA applications.

form those in FT-CORBA. However, logging mechanisms are disregarded as the servers are stateless. Fail-over mechanisms are built into the pure client and the intermediate-tier objects which are client of other servers so that each tier can itself trigger fail-over (rather than the pure client triggering the entire chain of fail-overs). With *stage-specific fail-over mechanisms*, when any object detects faults in its primary server replica, it can simply switch to the backup server replica, and re-issues the request. If this fail-over is successful, the end-to-end processing can be continued from the point of fail-over, without throwing away the processing completed by previous tiers.

## IV. Model of Nested Application

The purpose of our model is to estimate the worst-case fault-detection time for nested applications in order to assist developers in designing for their applications' real-time requirements. This model assumes that the application conforms to the architecture shown in Section III-A.

In this model, we have a client (tier 0)and a sequence of $n$ server objects (tiers 1 through $n$), as shown in Figure 3. The client invokes $S_1$, which processes the client's request and, in turn, invokes $S_2$, and so on, until $S_n$ processes the request and returns a response. Because each server is two-way warm passively replicated, if a fault occurs in a server, its preceding tier will detect the fault, and fail-over to the server's backup replica, which will then reprocess the request. Using one backup, we can tolerate a single crash fault of a server (with more backups, we can, of course, tolerate more faults). We assume that we are operating in a distributed asynchronous system where (i) the pure client does not fail (and is, therefore, not replicated), (ii) there are no correlated faults in the system, and (iii) the objects are deterministic.

We use the following symbols, some of which are illustrated in Figure 3. Unless mentioned otherwise, $i \geq 1$.

$n$ = number of server tiers ($i.e.$, excluding tier 0),
$X_i$ = processing time used in tier $i$,
$S_i$ = server object in tier $i$,
$m$ = tier where fault occurs, $m \geq 1$,
$t_i$ = nested-operation (round-trip) time measured at tier $i-1$ (including any reprocessing time after fail-over),
$d_i$ = time for tier $i-1$ to detect the fault in tier $i$.

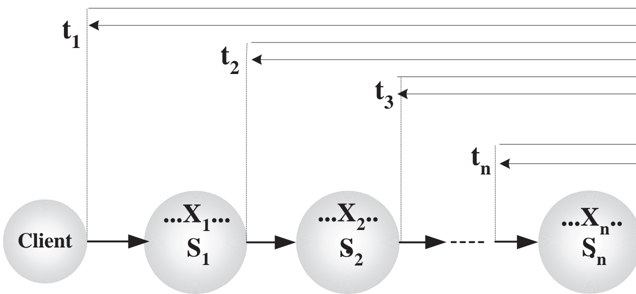


Fig. 3. Model of a nested CORBA application.

We define the following two tier-independent constants for the interaction between tier $i-1$ and tier $i$:

$T_r$ = round-trip time for a correct response (excluding the processing time for the invocation),
$T_e$ = round-trip time for an exceptional response (COMM_FAILURE),
$T_f$ = fail-over to the backup replica of tier $i$.

We define $st$ to be the stage of end-to-end processing when the fault is detected. For an application with $n$ tiers, $st$ can vary from 0 to $n$, depending on the status of the nested operation. For example, with three tiers, if the fault occurs in tier 1 ($m = 1$), $st$ has one of four possible values:

0: Before the client invokes $S_1$,
1: $S_1$ completes, and is about to invoke $S_2$,
2: $S_2$ completes, and is about to invoke $S_3$,
3: $S_3$ completes, and is about to respond to $S_2$.

### A. Determining the Worst-Case Performance

Since the worst-case performance of the system in the presence of a fault is of the most interest to real-time systems, we focus on estimating the worst-case fault-detection time (worst-case $d_m$ for a given $m$) and worst-case nested-operation time (worst-case $t_1$).

For a fixed $n$ and $m$, the values of $d_m$ and $t_1$ increase with $st$, as seen from the following argument. There might be some amount of time spent in processing beyond tier $m$ before the fault is detected by tier $m-1$. If the nested operation has not proceeded beyond $m$, $i.e.$, $st < m$, we do not need to throw away any of the processing. However, if $st \geq m$, any processing that occurs at, or beyond, tier $m$ has to be thrown away, and has to be re-done after fail-over to $m$'s backup. This time spent from tier $m$ to stage $st$ represents the "lost" processing-time under a fault. As $st$ becomes greater than $m$, the amount of "lost" processing-time and the amount of reprocessing increase; therefore, the nested-operation time $t_1$ (which includes the reprocessing) and the fault-detection time $d_m$ increase. The worst-case $d_m$ and the worst-case $t_1$ occur when $st = n$.

One of the factors in worst-case $d_m$ is the lost-processing time, which is bounded as follows:

$$\text{Lost processing time} \leq \sum_{i=m}^{n} X_i \tag{1}$$

Other factors affecting the worst-case $d_m$ are (i) the time $T_e$ to receive the COMM_FAILURE exception, and (ii) the sum of the round-trip time from tier $m$ to tier $n$ (excluding the processing time at tiers $m$ through $n$). The total round-trip time in (ii) is given by:

$$\text{Total round-trip time} = (n-m)T_r \tag{2}$$

The time, $d_m$, for the $(m-1)$st tier to detect the fault in the $m$th tier can be bounded as follows.

$$d_m \leq \sum_{i=m}^{n} X_i + T_e + (n-m)T_r \tag{3}$$

For a given $n$, the worst-case $t_1$ over all possible values of $m$ occurs when $m = 1$. Factors affecting the worst-case $t_1$ include (i) the worst-case fault-detection time $d_1$, (ii) the fail-over time ($T_f$) to tier 1's backup replica, and (iii) the reprocessing time, which includes the sum of the round-trip times ($nT_r$) and the processing times ($\sum_{i=1}^{n} X_i$) from tiers 1 through $n$. Expanding this and using Equation 3,

$$t_1 \leq T_e + T_f + (2n-1)T_r + 2\sum_{i=1}^{n} X_i \qquad (4)$$

## V. Empirical Validation

To study the characteristics of nested applications, and to validate the model of Section IV, we employ a multi-tier CORBA application with real-time fault-tolerant support conforming to the architecture of Section III-A. Our experimental test-bed uses the CORBA implementation provided by ACE 5.2.4 and TAO 1.2.4 [6] over Linux. The application objects, ACE and TAO were compiled with gcc 2.96.

Our test-bed consists of a client and five server objects, with configurable number of tiers, $i.e.$, the value of $n$ can be varied from 1 to 5, as needed, for each experiment. Each tier processes the invocation for 40 ms before invoking the next tier. We inject faults by killing a server replica at a time, in isolation. We refrain from injecting subsequent faults into the system while it is recovering. The objects in the application are distributed across four processors: one (1.4 GHz Pentium 4) for the client, a second (550 MHz Pentium 3) for the Naming Service, and the third and fourth processors (550 MHz Pentium 3 and 1.4 GHz Pentium 4) hosting replicas of all the tiers. Wherever possible, the primary replicas were located on the faster processors. All of the machines are connected by a 100 Mbps Ethernet.

### A. Experimental Results

For a fixed number $n$ of tiers, we inject a fault 100 times in tier $m$ ($1 \leq m \leq n$) for each possible stage $st$ ($0 \leq m \leq n$) of processing the nested operation. Note that $n$ itself can vary from 1 to 5, not within the same test-run, of course. Thus, a test-run consists of a fixed $n$, a fixed $m$ and a fixed $st$.

The data that we collect include the round-trip nested-operation times $t_i$ ($0 \leq i \leq n$) both in the fault-free case, and in the event of a fault. The fault-free data serve as our reference measurements. The data collected for each test-run is averaged over the 100 runs. We observed that the average round-trip time in the fault-free case increases with the value of $n$ (this result is rather obvious, and is, therefore, not shown in the graphs in this paper). The remainder of our data analysis focus on the experiments where faults were injected.

**Validating our Assumptions.** Figure 4 shows the round-trip time $t_1$ for $n = 5$ tiers for different values of $m$ (1 through $n$), averaged over all possible values of $st$ for each $m$. This graph shows that, when the fault occurs in an earlier tier, the application needs more time to reprocess the request than when the fault occurs in a later tier. Our experimental results bear out one of the assumptions in our model, namely, the fact the worst-case $t_1$ occurs when $m = 1$.

Figure 4 reveals the dependency on $m$, but not on $st$. Our model postulated that the greater that $st$ becomes over $m$, the more processing time we lose once the fault occurs; therefore, the round-trip time $t_1$ in the presence of a fault should increase as $(st - m)$ increases. This dependency on the value of $st$ is demonstrated by the results shown in the graph of Figure 5. Each line of this graph shows the average round-trip time $t_1$ when the fault occurs in a specific tier $m$, and the nested operation is in different stages $st$.

**Validating our Model.** We measured the constants $T_r$, $T_e$ and $T_f$ over multiple runs, and used the highest value to validate the worst-case model in Section IV-A. These values are $T_r = 1827$ $\mu$s, $T_f = 1307$ $\mu$s and $T_e = 2207.99$ $\mu$s. In addition, the worst-case $X_i = 43$ ms. Using these values in Equation 3 and Equation 4, we obtained our predicted worst-case bounds for $d_1$ and $t_1$, respectively. In all of the results discussed below, $m = 1$ and $st = n$ were fixed in order to measure the worst-case performance. Figure 6 shows the comparison between our calculated/theoretical $d_1$ bound and the measured values of $d_1$ for a fixed $n = 5$. As we can see, our calculated bounds are valid, except in the case of the abnormal "spikes" in the graph. Figure 7 shows the comparison between

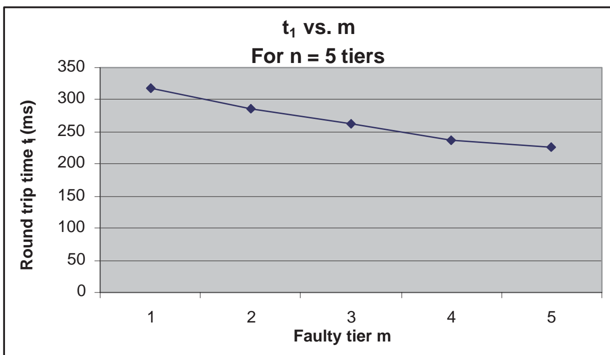


Fig. 4. Round-trip time $t_1$'s dependency on the tier $m$ where the fault occurs.

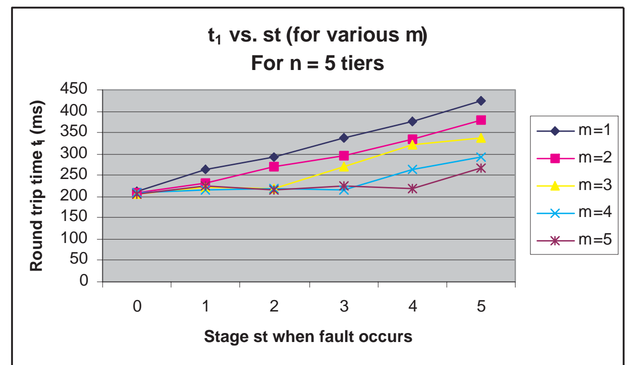


Fig. 5. Round-trip time $t_1$'s dependency on the stage $st$ when the fault occurs at specific values of $m$.

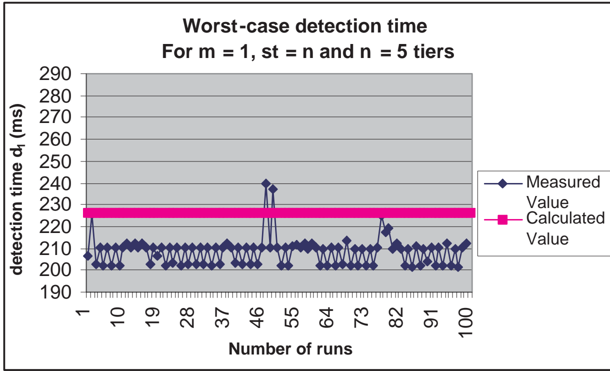


Fig. 6. Measured values vs. calculated values of worst-case detection time $d_1$ when $n = 5$.

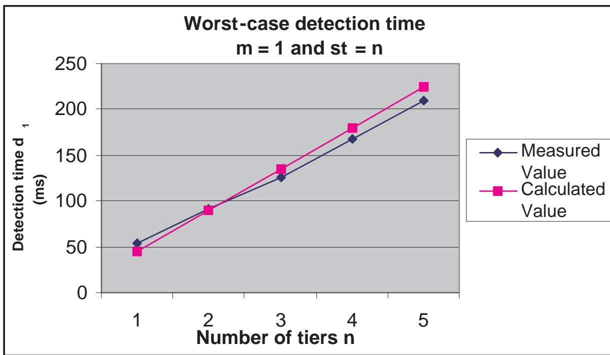


Fig. 7. Measured values vs. calculated values of worst-case detection time $d_1$ for various $n$

our theoretical $d_1$ bound and our experimental measurements with the varying number $n$ of tiers in the application. Figure 8 shows the comparison between our theoretical $t_1$ bound and our experimental measurements with the varying number $n$ of tiers in the application. In both Figure 7 and Figure 8, our worst-case theoretical values clearly bound the experimental measured ones, except when $n = 1$. We discovered that, when $n = 1$, the average $X_i$ is higher than the worst-case $X_i$ due to the presence of other unrelated tasks on the same processor, leading to this transient, unrepresentative behavior.

## VI. Related Work

There have been several non-middleware systems that have looked at some combination of real-time and fault tolerance for specific environments. In this section, we focus solely on current CORBA-based approaches. To the best of our knowledge, none of these systems address the fault-detection and fail-over issues when nested applications are considered.

Fault tolerant features have been added to the real-time CORBA implementation, TAO. The work adopts the semi-active replication style pioneered by Delta-4/XPA [1] in order to provide some guarantees of fault-tolerant determinism. The implementation currently supports single-threaded applications. The Real-time Object-oriented Adaptive Fault Tolerance

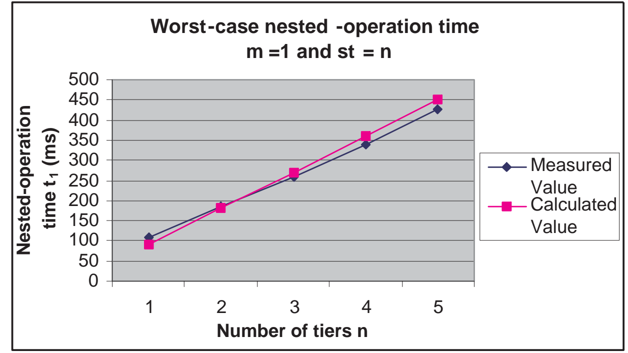


Fig. 8. Measured values vs. calculated values of worst-case round-trip time $t_1$ for various $n$

Support (ROAFTS) architecture [7] is designed to support the adaptive fault-tolerant execution of both process-stuctured and object-oriented distributed real-time applications. ROAFTS considers those fault tolerance schemes for which recovery time bounds can be easily established, and provides quantitative guarantees on the real-time fault tolerance of these schemes. A prototype has been implemented over the CORBA ORB, Orbix, on Solaris.

## VII. Conclusion

We explore the performance and the predictability of nested CORBA applications when faults occur in different tiers, and at different stages. We also develop a theoretical model to estimate the worst-case fault-detection time and the worst-case nested operation time for such applications. From our experimental measurements, we conclude that our model provides us with worst-case performance bounds that are accurate enough to enable the scheduling of real-time CORBA applications in the presence of faults and recovery. Our ongoing research includes the investigation of strategies for reducing the observed worst-case performance, and for tightening the theoretical bounds.

## References


[1] P. Barrett, P. Bond, A. Hilborne, L. Rodrigues, D. Seaton, N. Speirs, and P. Verissimo. The Delta-4 Extra Performance Architecture (XPA). In *Proceedings of the Fault-Tolerant Computing Symposium*, pages 481–488, Newcastle, UK, June 1990.

[2] P. Narasimhan. Trade-offs between real-time and fault-tolerance for middleware applications. In *Workshop on the Foundations of Middleware Technologies*, Irvine, CA, November 2002.

[3] Object Management Group. The Common Object Request Broker: Architecture and specification, 2.6 edition. OMG Technical Committee Document formal/2001-12-01, December 2001.

[4] Object Management Group. Fault tolerant CORBA. OMG Technical Committee Document formal/2001-09-29, September 2001.

[5] Object Management Group. Real-time CORBA. OMG Technical Committee Document formal/2001-09-28, September 2001.

[6] D. C. Schmidt, D. L. Levine, and S. Mungee. The design of the TAO real-time Object Request Broker. *Computer Communications*, 21(4):294–324, April 1998.

[7] E. Shokri, P. Crane, K. H. Kim, and C. Subbaraman. Architecture of ROAFTS/Solaris: A Solaris-based middleware for real-time object-oriented adaptive fault tolerance support. In *Proceedings of the Computer Software and Applications Conference*, pages 90–98, Vienna, Austria, Aug. 1998.