Towards Real-time Fault-Tolerant CORBA Middleware

Aniruddha S. Gokhale, Balachandran Natarajan
Douglas C. Schmidt
{gokhale,bala, schmidt}@isis-server.isis.vanderbilt.edu
Institute for Software Integrated Systems
Vanderbilt University
Box 1829, Station B
Nashville, TN 37235

Joseph K. Cross
joseph.k.cross@lmco.com
Lockheed Martin Tactical Systems
PO Box 64525, M S U2N29
St. Paul, MN 55164-0525

Abstract

An increasing number of applications are being developed using distributed object computing (DOC) middleware, such as CORBA. Many of these applications require the underlying middleware, operating systems, and networks to provide dependable end-to-end quality of service (QoS) support to enhance their efficiency, predictability, scalability, and reliability. The Object Management Group (OMG), which standardizes CORBA, has addressed many of these application requirements individually in the Real-time CORBA (RT-CORBA) and Fault-tolerant CORBA (FT-CORBA) specifications. Though the implementations of RT-CORBA are suitable for mission-critical commercial or military distributed real-time and embedded (DRE) systems, the usage of FT-CORBA with RT-CORBA implementations are not yet suitable for systems that have stringent simultaneous dependability and predictability requirements.

This paper provides three contributions to the study and evaluation of dependable CORBA middleware for performance-sensitive DRE systems. First, we provide an overview of FT-CORBA and illustrate the sources of unpredictability associated with conventional FT-CORBA implementations. Second, we discuss the QoS requirements of an important class of mission-critical DRE systems to show how these requirements are not well served by FT-CORBA today. Finally, we empirically evaluate new dependability strategies for FT-CORBA that can help make the use of DOC middleware for mission-critical DRE systems a reality.

Keywords: CORBA, Fault-Tolerant CORBA, Real-time CORBA, DRE systems

1 Introduction

Commercial-off-the-shelf (COTS) components based on distributed object computing (DOC) middleware, such as the Common Object Request Broker Architecture (CORBA) [1], are increasingly used to develop a wide variety of distributed real-time embedded (DRE) systems, such as commercial avionics systems, military combat systems, and supervisory control and data acquisition (SCADA) systems. CORBA is a DOC middleware standard defined by the Object Management Group (OMG) that allows clients to invoke operations on remote objects without concern for where the objects reside or what language the objects are written in [2]. In addition, CORBA shields applications from non-portable details related to the OS/hardware platform they run on or the communication protocols and networks used to interconnect distributed objects. These features make CORBA well suited to provide the core communication infrastructure for DRE systems and applications.

DRE systems demand dependable quality of service (QoS) support from their middleware, including efficiency, predictability, scalability, and reliability. The CORBA [1] standard addresses many of these challenges via the following specifications:

Real-time CORBA (RT-CORBA). RT-CORBA [3] provides capabilities to ensure predictable behavior end-to-end for requests that traverse from one object to another. To deliver these capabilities *vertically* (*i.e.*, network interface ↔ application layer) and *horizontally* (*i.e.*, peer-to-peer), RT-CORBA defines standard interfaces and QoS policies that allow applications to configure and control the following types of resources:

- *Processor resources* via thread pools, priority mechanisms, intra-process mutexes, and a global scheduling service for real-time applications with fixed priorities,
- *Communication resources* via protocol properties and explicit bindings to server objects using priority bands and private connections, and
- *Memory resources* via buffering requests in queues and bounding the size of thread pools.

Although RT-CORBA is oriented towards applications with hard real-time requirements, such as process control and weapons systems, it also supports applications with stringent soft real-time requirements, such as telecommunication call processing. A comprehensive overview of RT-CORBA appears in [4]. Implementations of RT-CORBA including TAO [5] and ORBExpress (`www.ois.com`) have been evaluated positively [6] for their suitability to DRE systems that require end-to-end predictability and real-time assurance. RT-CORBA ORBs have also been used to

- Improve situation awareness for prosecuting time-critical targets [7], such as mobile missile launchers and tanks, and
- Control avionics mission computing application processing in real-time [8, 9].

Fault-tolerant CORBA (FT-CORBA). FT-CORBA [1] defines services and strategies to enhance the dependability of CORBA applications. The fault tolerance mechanism used by FT-CORBA to detect and recover from failures is based on *entity redundancy* – in particular, the replication of CORBA objects. In general, research on fault tolerance for CORBA ORBs and its applications can be divided into the following three strategies [10]:

- The *integration* strategy, where the ORB is modified to provide the necessary fault tolerance support and the extent of the modifications depends on the level of fault-tolerance support that is being added. Orbix+Isis [11] and Electra [12] are examples of the integration strategy.
- The *interception* strategy, where requests made by client objects are captured *externally* to the ORB via an OS-level interceptor [13], which can enhance the application by providing support to tolerate faults. The Eternal system [14], and the AQuA framework [15] are examples of the interception strategy.
- The *service* strategy, where a set of interfaces and objects are defined as a CORBA service that provides the policies and mechanisms for delivering fault tolerance to applications. The Distributed Object-Oriented Reliable Service (DOORS) [16, 17, 18] and Object Group Service (OGS) [19] are examples of this strategy.

The FT-CORBA implementations outlined above also leverage many patterns and protocols pioneered from earlier R&D efforts, including NewTop [20], Interoperable Replication Logic (IRL) [21], and FRIENDS [22].

Unfortunately, mission-critical DRE systems, such as shipboard combat control systems and avionics mission computing systems, that require support for multiple QoS properties simultaneously are not yet well supported by FT-CORBA for the reasons explained shortly. These mission-critical DRE systems are typified by the following characteristics:

- **Stable applications** – Most DRE systems have a longer life than their commercial counterparts, which requires that the infrastructure for DRE systems provide stable interfaces [23]. This in turn provides DRE systems with the flexibility to modify the underlying infrastructure as long as the interfaces remain compatible.
- **End-to-end timeliness and dependability requirements** – DRE systems have stringent latency and dependability requirements. Latency requirements commonly bound the time from the occurrence of an external event until the system delivers an externally observable response, whereas dependability requirements are often expressed as a probabilistic guarantee that the requirements will be met.
- **Heterogeneity** – DRE systems often run on a wide variety of computing platforms that are interconnected by different types of networking technologies. The efficiency of execution of the different infrastructure components on which the DRE systems operate varies with the type of computing platform and its interconnection technology.

Simultaneously providing dependability and predictability properties for the class of DRE systems outlined above is hard since the combination of these properties is often in conflict, as explained in Section 3. For example, FT-CORBA implementations can spend an unpredictable amount of time detecting and recovering from faults. This in turn conflicts with the bounds on latency for message invocation since FT-CORBA must account for the time spent on fault detection and recovery. Consequently, providing both these QoS requirements simultaneously requires a careful blend of protocols, patterns [24], and design constraints and tradeoffs, which transcends the present capabilities of COTS DOC middleware.

Our prior research on CORBA middleware has explored the efficiency, predictability, scalability and dependability aspects of ORB endsystem design, including static [25] and dynamic [26] scheduling, event processing [9], I/O subsystem [27] and pluggable protocol [28] integration, synchronous [29] and asynchronous [30] ORB Core architectures, systematic benchmarking of multiple ORBs [31], and optimization principle patterns for ORB performance [32]. This paper extends our previous work by focusing on the following dimensions in the ORB endsystem design space:

- Identifying key aspects of CORBA implementations that deliver both real-time and fault tolerant properties simultaneously
- Evaluating the suitability of FT-CORBA as the dependability infrastructure for DRE systems and
- Designing and empirically evaluating architectural enhancements to FT-CORBA for use in DRE systems.

The remainder of this paper is organized as follows: Section 2 summarizes the FT-CORBA specification; Section 3 describes key challenges that must be resolved when designing dependable DRE systems using CORBA; Section 4 proposes a replication style that addresses the key challenges outlined in Section 3; Section 5 evaluates and compares the technique described in Section 4 with existing FT-CORBA strategies; Section 6 compares related work with our research; and Section 7 presents concluding remarks and outlines our future research directions.

2 Overview of Fault Tolerant CORBA

The Fault Tolerant CORBA (FT-CORBA) [1] specification defines a standard set of interfaces, policies, and services that provide robust support for applications requiring high reliability. The fault tolerance mechanism used in FT-CORBA to detect and recover from failures is based on *entity redundancy*. Since FT-CORBA is a DOC middleware standard, the redundant entities are replicated CORBA objects. This section presents an overview of the FT-CORBA specification.

2.1 Overview of the FT-CORBA Architecture

Fault tolerance for CORBA objects is achieved via *replication*, *fault detection*, and *recovery*. Replicas of a CORBA object are created and managed as a "logical singleton" [24] composite object, which allows greater flexibility in configuration management of the replicas. Such a collection of replicas is called an *object group*. Figure 1 illustrates the key components in the FT-CORBA architecture. All components shown in the fig-

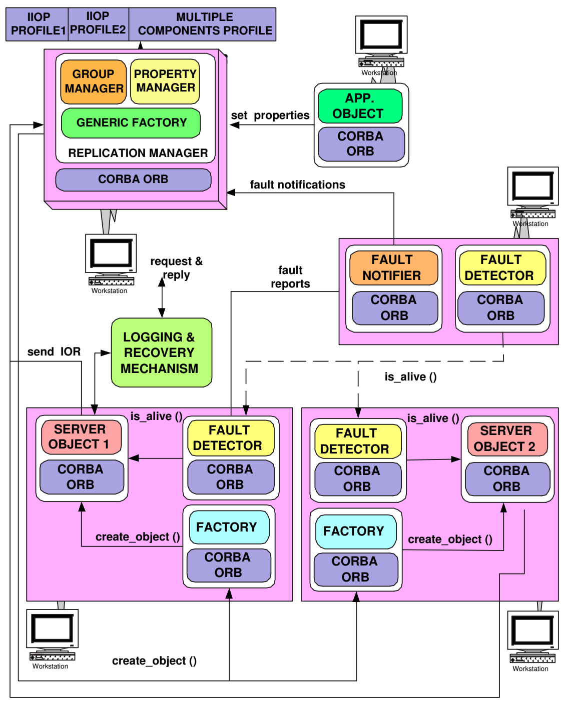

Figure 1: **The Architecture of FT-CORBA**

ure are implemented as standard CORBA objects, *i.e.*, they are defined using CORBA IDL interfaces and implemented using servants that can be written in standard programming languages, such as Java, C++, C, or Ada. The functionality of each component is described below. Sidebar 1 explains the requirements placed by the FT-CORBA standard on the underlying CORBA ORB middleware.

2.1.1 Interoperable Object Group References (IOGRs)

CORBA standardizes the format of interoperable object references (IOR) used for the individual replicas. An IOR is a flexible addressing mechanism that identifies a CORBA object [1]. Additionally, FT-CORBA takes advantage of CORBA's notion of a multi-profile IOR, where each profile contains a path to the location of the object, and defines an IOR for composite objects called the *interoperable object group reference* (IOGR), which is illustrated in Figure 2. In FT-CORBA,

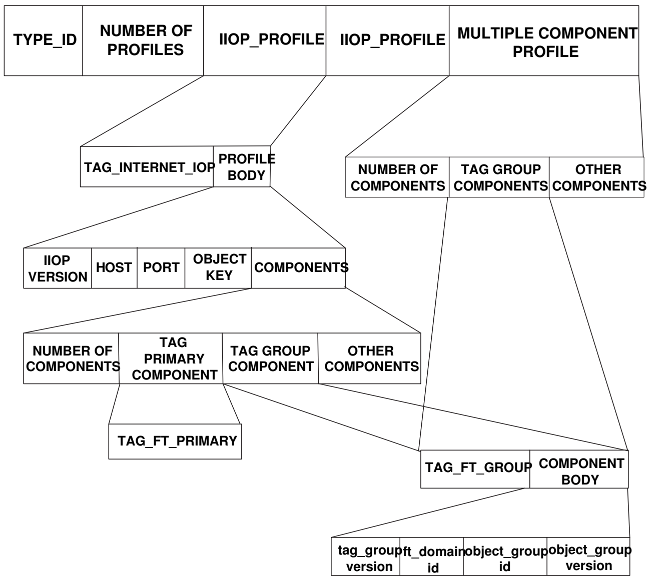

Figure 2: **Example of an IOGR**

each profile is interpreted to be the replica of the object. An IOGR contains multiple TAG_INTERNET_IOP profiles, any of which can be used to reach the server object group. The TAG_FT_GROUP component is contained in every profile of the reference. The TAG_FT_PRIMARY component is contained in only one profile of the reference.

FT-CORBA servers can publish IOGRs to clients, which then use these IOGRs to invoke operations on servers. The client ORB transmits the request to the appropriate server object that handles the request. The client application need not be aware of the existence of server object replicas. If a server object fails, the client ORB iterates through the object references contained in the IOGR until the request is handled successfully by a replica object. The IOGR is considered invalid only if all server replicas fail, in which case an exception is propagated to the client application. The FT-CORBA specification [1] contains the details of the exceptions that are returned and the mechanisms available to populate the client with the new IOGR when the old one becomes stale.

2.1.2 Replication Manager

The *ReplicationManager* is responsible for managing replicas and contains the following three components:

1. PropertyManager. This manager allows properties of an object group to be selected. Common properties include the replication style, membership style, consistency style, and initial/minimum number of replicas. Replication styles supported by FT-CORBA include the following:

- COLD_PASSIVE, where the replica group contains a single primary replica that responds to client messages. If the primary fails, then an idle replica is selected and the state of the failed primary is loaded into that replica, which then becomes the new primary.
- WARM_PASSIVE, same as COLD_PASSIVE, except that the state of the primary is periodically loaded into the backup replicas, so that only a (hopefully minor) update to that state will be needed for failover.
- ACTIVE, where all replicas are primary and handle client requests independently of each other. FT-CORBA uses reliable multicast group communication [20, 33, 34, 19] to provide ordered delivery of messages and to maintain state consistency among all replicas. The infrastructure sends a single reply to the client by detecting and suppressing duplicate replies generated by multiple members of the object group.
- ACTIVE_WITH_VOTING, which is a planned extension to FT-CORBA where replicas behave similarly to ACTIVE style, but the middleware selects a reply by conducting an *election* among the multiple replies, and dispatches the selected reply to the client.

Either the FT-CORBA infrastructure or applications can control membership of an object group and the data consistency of the group members. FT-CORBA standardizes both application-controlled and infrastructure-controlled membership and consistency styles.[1]

2. GenericFactory. The `ReplicationManager` uses the `GenericFactory` to create object groups and individual members of an object group for the infrastructure-controlled membership style.

3. ObjectGroupManager. For the application-controlled membership style, applications use the `ObjectGroupManager` interface to create, add, or delete members of an object group.

[1]FT-CORBA does not standardize the actual implementation details for application-controlled membership or consistency styles.

2.1.3 Fault Detector and Notifier

`FaultDetectors` are CORBA objects responsible for detecting faults via either a *pull-based* or a *push-based* mechanism. A *pull-based* monitoring mechanism polls applications periodically to determine if their objects are "alive." FT-CORBA requires application objects to implement a `PullMonitorable` interface that exports an `is_alive()` operation. The FT-CORBA specification does not limit the number or arrangement of `FaultDetectors` in a domain, which is explained in Section 2.1.5. For example, a large system spanning multiple hosts and supporting many objects in a hierarchical structuring of `FaultDetectors` is more scalable and efficient.

A `FaultDetector` reports the faults it identifies to a `FaultNotifier`. In turn, a `FaultNotifier` propagates these notifications to the `ReplicationManager`, which performs recovery actions. Other applications in the system that are interested in monitoring fault activity can also register with the `FaultNotifiers` to receive fault notifications.

Complex applications can provide `FaultAnalyzers` to expand, correlate, condense, and analyze fault reports. The functionality provided by `FaultAnalyzers` is usually platform- and application-specific. For example, a sequence of fault reports can be correlated to identify a single failure condition.

2.1.4 Logging and Recovery

Applications that select the application-controlled consistency style are responsible for their own failure recovery. For applications that select the infrastructure-controlled consistency style, however, FT-CORBA defines a logging and recovery mechanism. This mechanism intercepts and logs CORBA GIOP messages from client objects to servers. Figure 3 illustrates how the logging mechanism operates during normal operation. As part of the recovery action after a fault occurs, the

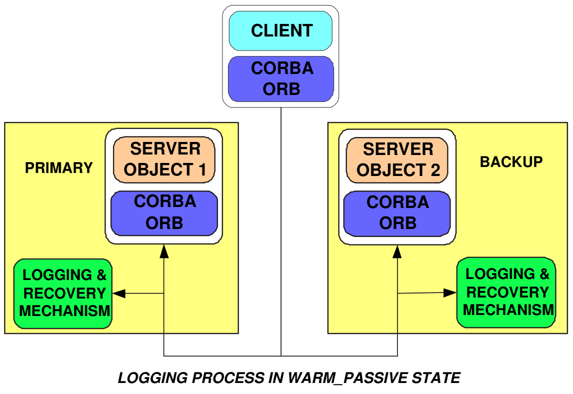
CLIENT
CORBA ORB
PRIMARY
SERVER OBJECT 1
CORBA ORB
LOGGING & RECOVERY MECHANISM
SERVER OBJECT 2
CORBA ORB
BACKUP
LOGGING & RECOVERY MECHANISM
LOGGING PROCESS IN WARM_PASSIVE STATE

Figure 3: **Operation of the FT-CORBA Logging Mechanism**

messages that are recorded are played back to the new primary so its state is consistent with that of the old primary before the failure occurred.

Figure 4 illustrates how the recovery mechanism applies messages from the log to the replica to synchronize it with the current state. In the WARM_PASSIVE replication style, backup

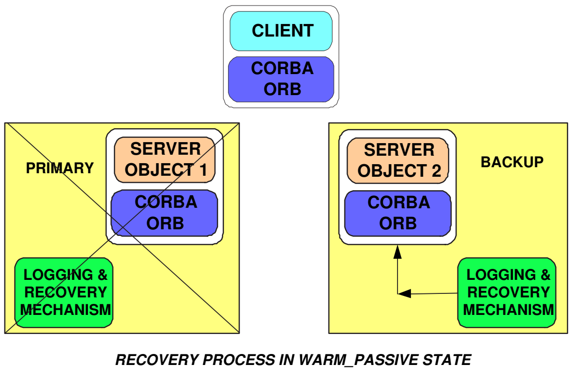
CLIENT
CORBA ORB
PRIMARY
SERVER OBJECT 1
CORBA ORB
LOGGING & RECOVERY MECHANISM
SERVER OBJECT 2
CORBA ORB
BACKUP
LOGGING & RECOVERY MECHANISM
RECOVERY PROCESS IN WARM_PASSIVE STATE

Figure 4: **Operation of the FT-CORBA Recovery Mechanism**

members in the object group should receive state updates at constant intervals of time during normal operation. When a backup member is promoted to a primary after failure, however, the FT-CORBA recovery mechanism only applies recent state updates on the failed primary to the backup member, after the last successful update. For an object group configured with the COLD_PASSIVE replication style, a backup is created and the recovery mechanism applies the complete log to the backup.

After all replicas are consistent, the FT-CORBA recovery mechanism then reinvokes the operations that were made by the client, but which did not execute due to the primary replica's failure. These FT-CORBA logging and recovery mechanisms ensure that failovers are transparent to applications.

2.1.5 Fault Tolerance Domains

To manage large and complex distributed systems, the FT-CORBA standard defines *fault tolerance domains*, which are designed to allow applications to scale to arbitrary sizes. A single fault tolerance domain consists of one or more hosts and one or more object groups, as illustrated in Figure 5. In addition, hosts can be part of several domains simultaneously. Complex, large-scale applications consist of several object groups that often span one or more fault tolerance domains. All object groups within a single fault tolerance domain are managed by a single logical domain-specific `ReplicationManager`. Objects without knowledge of FT-CORBA and/or that reside outside the domain can communicate with replicated objects inside a domain via a `Gateway`. The `Gateway` can use a group communication protocol, such as Totem [33], to multicast the IIOP call from the client outside the domain.

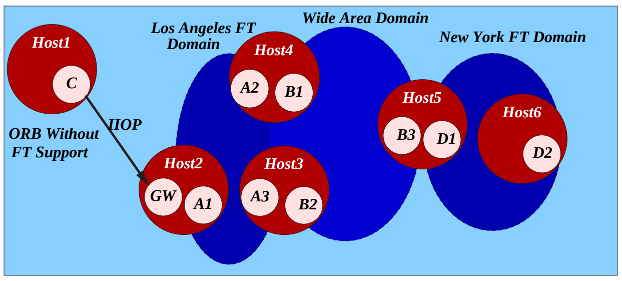

Figure 5: **FT-CORBA Domains**

3 Limitations of FT-CORBA for DRE Systems

The majority of computational cycles today are expended to control DRE systems, including commercial and military aircraft and satellites, automobile engines, chemical and manufacturing plants, and hospital patient monitoring equipment. Due to constraints on weight, power consumption, memory footprint, and performance, DRE systems are harder to develop, maintain, and evolve than mainstream desktop and enterprise software. During the past decade, a substantial amount of R&D effort has focused on developing distributed object computing (DOC) middleware as a means to simplify the development and reuse of DRE systems.

The characteristics of DRE systems described in Section 1 motivate the integration of implementations of RT-CORBA and FT-CORBA as the infrastructure for DRE systems. This DOC middleware provides open standard interfaces that simplify the development of DRE systems requiring dependability and predictability. As discussed below, however, their combined use in today's ORBs lacks certain features and have semantically incompatible strategies that make them unsuitable for important types of DRE systems. The remainder of this section describes the challenges associated with integrating the RT-CORBA and FT-CORBA specifications to meet the QoS requirements of DRE systems.

3.1 Challenge 1: Unpredictable and Expensive Replication Strategies

Context. Application objects use replication to achieve transparent fault tolerance. As discussed in Section 2.1.2, FT-CORBA specifies the COLD_PASSIVE, WARM_PASSIVE, ACTIVE, and ACTIVE_WITH_VOTING replication styles to tolerate

Sidebar 1: Requirements of the FT-CORBA Standard

FT-CORBA imposes the following requirements on the underlying CORBA ORB middleware:

Preserving the CORBA object model. For the infrastructure-controlled consistency style, the behavior of a replicated object should appear as though it is a single non-replicated CORBA object.

Enhancements to the CORBA object reference model. FT-CORBA defines three new *tagged components* into the object reference model to denote multiple components in a replicated object reference. The standard mandates that ORBs not supporting FT-CORBA should be able to handle such object references. In addition, objects hosted by such ORBs should be able to invoke operations on these multiple profile object references.

No single point of failure. FT-CORBA is designed to prevent single points of failure within a distributed system. As a result, each component described above must itself be replicated and mechanisms provided to deal with potential failures and recovery.

Periodic Fault detection and notification. The FT-CORBA standard mandates periodic fault detection and notification of faults to the `ReplicationManager` and subsequent recovery from failures.

Transparent failovers. Recovery from failure and the number of replicas in an object group should be transparent to clients making requests to the object group. Moreover, as mentioned earlier, client applications should not need to distinguish between sending requests to a replicated object or a non-replicated object.

Transparent client redirection and reinvocation. As explained earlier, the standard defines an IOGR that a client ORB uses to send requests to object group replicas. If a failure occurs when a client communicates with an IOR within the IOGR, the client ORB redirects the request to other IORs within the IOGR. The client ORB systematically reinvokes the request until the request succeeds. This redirection and reinvocation of requests should be transparent to client applications. Sidebar 2 explains how the *at-most-once* semantics of CORBA requests are maintained.

faults transparently. With the exception of COLD_PASSIVE and WARM_PASSIVE, these replication styles require that replicas maintain consistent state after every invocation.

Problem. In COLD_PASSIVE and WARM_PASSIVE systems, the recovery time needed to switch to a backup replica can be unacceptably high for DRE systems with stringent timing constraints, as shown in Section 5. Likewise, in ACTIVE replication the cost associated with providing totally ordered reliable multicast and the time needed to synchronize via proprietary group communication mechanisms can be unacceptable [35].

Using an ACTIVE replication for applications based on “push-pull” architectures, such as the CORBA Event Service [9], can introduce non-determinism and unpredictability when trying to handle multiple events. Moreover, the replicas

Sidebar 2: Preventing Multiple Invocations

To maintain the *at-most-once* semantics of the CORBA object model, the FT-CORBA standard defines a REQUEST service context that a client ORB includes in application requests. This service context includes a unique *client ID* for the client, a *retention ID*, and an *expiration time* provided by the client. The client ID and retention ID together uniquely identify a request. This mechanism is used by the server ORB to identify duplicate requests. For any duplicate request that has already been successfully serviced before, the server ORB sends identical replies as before from the log of requests and corresponding replies it maintains. The expiration time is used to determine the amount of time that a server ORB should maintain the log for a request and its corresponding reply, if any.

FT-CORBA also defines a GROUP_VERSION service context that a client ORB can send with a request. This service context includes the group version number of the replica group to which it sends a request. A server ORB uses this information to determine if its associated client ORB has an obsolete IOGR for the server object group. If the IOGR is obsolete, the server ORB sends a LOCATE_FORWARD_PERM message to the client ORB with the new IOGR.

in ACTIVE replication must behave identically and deterministically if they are used in DRE systems. Special negotiations (which impose high overhead) are needed to enforce the consistent execution order of messages among all the replicas. Preemption of requests, such as responding to an alarm condition, in ACTIVE replication can make the system non-deterministic, which can be avoided only via complex protocols that consume significant system resources.

FT-CORBA also requires strong replica consistency, which for ACTIVE replication means that all members of the object group must have the same state at the end of each method invocation. It is conceivable, however, that application objects required to be fault tolerant will also make outcalls, such as reading the real-time clock or responding to arbitrary external events. Under these circumstances, it is unlikely that consistent state can be maintained across the replicated objects, which suggests that a conforming implementation must severely constrain the ways in which objects can interact with their environment.

In Section 4 we describe the SEMI-ACTIVE replication style that provides the following benefits:

- Fast and predictable failure detection times
- Predictable state synchronization strategy that eliminates the need for protocols with high overhead
- No restriction placed on the ways in which an application can interact with its environment, and
- Can be standardized as an enhancement to the present FT-CORBA specification.

3.2 Challenge 2: Dealing with Semantic Incompatibilities Between RT-CORBA and FT-CORBA Features

Context. Requirements on DRE systems are commonly expressed in terms of external stimuli and responses. For example, consider a tactical display system that uses radars as sensors and that presents an operator with a graphical representation of the geographical area, including the present locations of moving objects, such as missiles or tanks. In such systems, a common requirement is *radar to glass in one second*, which means that at most one second may elapse between a radar pulse bouncing off the surface of a target and the corresponding observation being displayed to the operator.

Similarly, dependability requirements are often expressed in terms of the probability that one or few of the many requirements will fail to hold over a specific period of operation. DRE systems, such as the tracker outlined above, require simultaneous stringent QoS properties, including predictability and dependability, for their correct operation.

Problem. Although combining the power of RT- and FT-CORBA seems a promising approach, the requirements on DRE systems illustrate a significant semantic gap between

- End-to-end predictability and dependability requirements, such as 500 milli-second response times even in the presence of crash faults, and
- Capabilities, such as propagating priorities and replicating server objects.

The ability to engineer a good fault tolerant solution requires tradeoffs that may compromise a DRE system's ability to meet real-time deadlines, and vice versa. For example, a non-trivial amount of communication and processing costs are incurred to accomplish failover from one replica to another. This overhead presents a hard choice between accounting for failover in every message transmission versus preparing for time budgets to be violated during failover.

In general, the variation in performance and latency inherent in an elaborate FT solution is fundamentally antithetical to the predictable behavior required in a DRE system. The effect of choices made to support a requirement in one area can have complex and unforeseen consequences in the other. For DRE systems to leverage the advantages of open, standard interfaces, therefore, a solution that can mask the semantic incompatibility between FT-CORBA and RT-CORBA solutions is needed.

Schemes like [36, 37, 38] have been developed and deployed that address the conflict between real-time and fault-tolerance capabilities of non-CORBA systems. We propose and evaluate a strategy in Section 4 for CORBA systems that provides bounded fault-tolerance capabilities, while also maintaining the real-time properties essential to develop dependable DRE systems.

3.3 Challenge 3: Lack of Standards to Handle Byzantine and Partial Failures

Context. Partial failures are not uncommon in distributed systems. A component interacting with another component often cannot distinguish whether a delay in response is due to failure of the remote component or due to a slow or partitioned network. A system is considered reliable if it performs its designated function, even in the face of partial failures.

Problem. The FT-CORBA model for failure detection and recovery emphasizes a certain type of failure, namely *component failure*, which is also called *crash failure*. In this type of failure the individual component ceases all interactions with its environment. The policies and detection mechanisms in FT-CORBA, such as the use of heartbeats and timeouts, acknowledge this limited view implicitly.

A more subtle form of failure is one where interactions among components cause the system to fail. For example, a corrupted component may request services more frequently than allowed for in the design, thereby denying other components access to critical resources. Heartbeats and timeouts cannot protect against this type of failure. Given the inability to detect this difference, the requirement that objects using ACTIVE replication maintain consistent state in a bounded time between method invocations may be infeasible to achieve in the general case.

Section 7 discusses the future work that is needed to resolve this challenge.

3.4 Challenge 4: Lack of Standard QoS Semantics

Context. For designers of DRE systems, the primary benefit of an open standard is the promise of a stable interface between the application and the services provided by the middleware, since this simplifies porting to a different service implementation. In particular, when a service is part of the infrastructure—and the infrastructure is built with COTS products—system designers are highly motivated to consider both the cost of upgrades and penalties associated with COTS obsolescence [23]. It is therefore important that DRE systems interact with the infrastructure through syntactically and semantically stable interfaces, such as those defined in the RT-CORBA and FT-CORBA specifications. CORBA also helps improve the portability and interoperability of applications built using such interfaces.

Problem. The RT-CORBA and the FT-CORBA specifications provide the mechanisms by which real-time and fault-tolerant behavior can be achieved for DRE systems. CORBA does not, however, guarantee that two implementations conforming to the RT-CORBA and FT-CORBA specification will provide equivalent QoS properties. For example, assume that a system implemented on ORB$_A$ (which is compliant with the RT-CORBA and FT-CORBA specifications) successfully meets the requirement that sensor data from the navigation subsystem be propagated to all interested recipients at 35 millisecond (ms) intervals, even when certain faults occur in the system. It is highly unlikely, however, that if ORB$_A$ were replaced by ORB$_B$, the 35 ms propagation interval will be maintained, even if ORB$_B$ also complies with the RT-CORBA and FT-CORBA specifications. To establish that the modified system continues to meet its requirement would require thorough testing, possible reengineering, and expensive re-certification.

In general, the absence of a QoS standard makes the existing RT-CORBA and FT-CORBA specifications inadequate for certain types of DRE systems. Moreover, the implementation freedom that arises from a purely functional interface specification means that performance characteristics of various compliant products may vary greatly. Such performance variations are problematic when ORB$_A$ is replaced by a different (yet still functionally compliant) ORB$_B$. For instance, one product may choose to use an inter-process communication (IPC) mechanism, such as shared memory, to communicate between objects collocated on the same computer, whereas another may pass the message through an IPC loopback device. These two mechanisms can produce radically different levels of performance. Replacing or updating an ORB may therefore require costly redesign, reimplementation, and revalidation of DRE systems.

Section 7 discusses the future work that is needed to resolve this challenge.

4 Overview of the SEMI-ACTIVE Replication Style

Section 3 described the challenges of using FT-CORBA and RT-CORBA together to build DRE systems. Though the challenges in Sections 3.3 and 3.4 must be addressed to build robust, reliable and long-running DRE systems, the challenges in Sections 3.1 and 3.2 should be addressed first, since they are more fundamental and form the basis by which the others can be addressed. This section presents the SEMI-ACTIVE replication style to address these challenges.

SEMI-ACTIVE replication is based on the European Delta-4 (XPA) architecture [36] (where this term was coined). It is designed to enhance active replication and passive replication styles to support predictable program execution and failover times, without incurring the overhead, unpredictability and non-determinism of the standard FT-CORBA strategies outlined in Section 2.1.2. This section describes our efforts to integrate SEMI-ACTIVE replication into FT-CORBA. Section 5 then compares SEMI-ACTIVE replication with existing replication styles supported by FT-CORBA.

Sidebar 3: An Overview of SEMI-ACTIVE Replication

The key features of the SEMI-ACTIVE replication style are outlined below:

- The replicas are arranged as a list of nodes, where each replica except the primary is connected through a connection-oriented transport-level connection to the one ahead in the queue.
- The list of replicas are created at server startup time.
- The replica at the head of the list is designated the primary.
- When the primary fails, the next secondary replica in the list is promoted to become the primary.
- Failures are detected by the next replica in the list when transport-level connections close. Connection closure, which is an event in the system, is used to detect failures. This is done by making the secondaries wait on open connections using an event demultiplexing mechanism.
- The promotion of the secondary to the primary is done by the secondary when it detects a failure.
- All client invocations to the primary are reliably multicast by the primary to the secondaries, such that secondaries consume messages in the same order that the primary consumes them, maintaining replica consistency.
- More generally, replica consistency can be maintained by reliably multicasting state synchronization messages to all replicas.
- The multicast protocol that is used for request invocation or state transfer needs to enforce message ordering. A simple reliable model that ensures data delivery alone is not sufficient.
- An ordered list of references is passed to the client, which must honor the order of the list.
- No additional threads are created for sending heartbeats or monitoring server objects.

Sidebar 3 describes the key features of this replication style. Figure 6 illustrates how the replicas are arranged to tolerate faults in SEMI-ACTIVE replication. The SEMI-ACTIVE replication style resolves the following challenges with the existing mechanisms described in Section 3.1 and 3.2:

- The efficient and predictable failure detections ensure faster and more predictable recovery times when compared to COLD_PASSIVE and WARM_PASSIVE replication.
- No need for additional protocols based on algorithms like priority based total ordered multicast [39], which enforce inter-replica coordination required to guarantee priority-based message ordering among replicas for the ACTIVE replication style. The complexity and overhead of these protocols is avoided by SEMI-ACTIVE replication.
- Reduced heartbeat and poll messages on the network since they are not used for detecting failures.

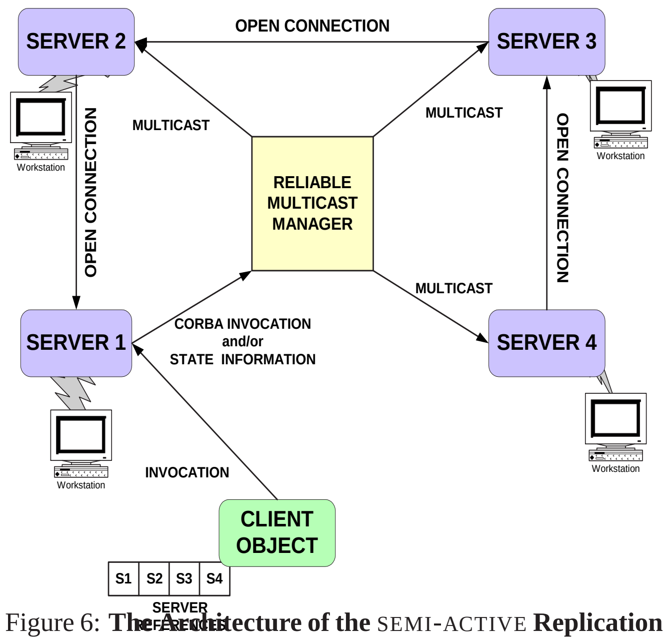

Figure 6: **The Architecture of the** SEMI-ACTIVE **Replication Style**

- The RT-CORBA mechanisms and policies can be used easily with this strategy.

Despite the advantages outlined above, SEMI-ACTIVE replication style has the following disadvantages:

- If a non-primary replica fails, the replica just following the failed replica could declare itself as a primary and wait for messages to be processed, which can potentially partition the list. To prevent partitioning, a remote token manager should disseminate tokens to replicas and promote them as primaries in a predictable manner.
- When a primary fails, the FT-CORBA model does not place any restriction on the client's choice of a backup from the list of replica references to make the invocation. The SEMI-ACTIVE replication style could restrict the client to use the list of references as an ordered list, which is not compliant with the FT-CORBA spec.
- As with all replication styles other than COLD_PASSIVE, applications must provide their own replica consistency mechanisms if their state changes due to operations other than regular CORBA requests.[2] Similarly with SEMI-ACTIVE replication, applications choosing to multicast state information instead of multicasting requests could use interceptors that pass state information to the multicast mechanisms. Application developers are responsible for installing these mechanisms. In other words, application developers have to program to this replication strategy.

Fortunately, the disadvantages outlined above do not affect the SEMI-ACTIVE replication style's predictability, which makes it a good candidate for use in DRE systems. Section 5 empirically evaluates key properties of this replication style.

5 Empirically Evaluating Semi-Active Replication

This section compares the performance of the SEMI-ACTIVE replication style described in Section 4 with the FT-CORBA replications styles discussed in Section 3. We also illustrate how the SEMI-ACTIVE replication style compares favorably with the FT-CORBA replication styles. Our experiments assume a single-failure model with no nested failures. The faults occuring in our experiments are assumed to be crash failures injected at random. We explain below the methodology we adopted for inducting crashes.

5.1 Comparing Semi-Active Replication with Warm_Passive Replication

Below, we describe the results of empirical benchmarking studies conducted to measure how well the SEMI-ACTIVE replication style compares to the WARM_PASSIVE replication style in providing real-time and fault-tolerance support to DRE systems. A key goal in conducting these benchmarks is to evaluate the predictability of the SEMI-ACTIVE replication style in terms of its

1. *Detection time* to recognize failures and
2. *Response time* required for clients to connect to a new primary after an existing primary fails.

To evaluate the SEMI-ACTIVE style, we built several tests that demonstrate specific use cases for these benchmarks. The tests were based on ACE [40, 41] and TAO [25], versions 5.2.2 and 1.2.2, respectively. The tests were run on a single endsystem – a 930 MHz Pentium III processor with 512 MB RAM running version 2.4.9 of the Linux kernel in the FIFO real-time scheduling class.[3] For the WARM_PASSIVE replication we use data collected from our previous benchmarking experiments in [18]. We compare only the trend in data and not the absolute numbers in the following sections.

[2]There is a class of DRE systems whose object state changes are triggered by occurrence of events, such as time triggers or alarm conditions, rather than exclusively from CORBA requests.

[3]Although we use TCP to benchmark the SEMI-ACTIVE replication style in our experiments, the stream control transmission protocol (SCTP) [42] and Myrinet [43] are more effective transports for DRE systems in practice since they provide better fault detection times and better real-time guarantees than TCP.

5.1.1 Comparing Failure Detection Time on the Server

Rationale. We define the *failure detection time* on the server as the time taken to detect a failure, which includes the time taken by the infrastructure to detect the failure of the primary. This value is important since it affects the time taken by the middleware infrastructure to detect and react to faults.

Methodology for SEMI-ACTIVE. To model the replicas, we used a server based on the Acceptor and Connector [41] framework components in ACE. The primary waits for input requests from the client on a specified port and sends a response to the client to mark the end of an invocation. In addition to waiting for connections from the client, the secondary also connects to the primary, as shown in Figure 6. The client establishes connections to all the replicas since this reduces the jitter due to connection establishment during failover.

A script creates the primary and all the other replicas. We then allow a client to connect to the replicas and invoke some random number of remote operations. At this point we invoke the primary object's `shutdown()` operation, which crashes the primary. This crash initiates a detection process in the secondary, which promotes itself to the primary and waits to receive invocations.

We measure the failure detection time as the time between the failure of the primary replica and the time when the secondary actually detects a failure. To measure the failure detection time, we recorded two time stamps:

- The first time stamp was recorded when the primary was killed.
- The second timestamp was recorded when the secondary detected that its connection to the primary was closed.

We conducted several iterations of this experiment by killing the primary replica at randomly selected times.

Methodology for WARM_PASSIVE. To model the replicas under WARM_PASSIVE replication we used the infrastructure provided by DOORS [16, 17, 18], which is our prototype implementation of the FT-CORBA spec. A manager program requests the `Replica Manager` to start the replicas. After the replicas are running, the `FaultDetector`s begin polling them at constant intervals of time. We then allow a client to connect to the primary replica and invoke a random number of remote operations. At this point, we invoke the server object's `shutdown()` operation, which initiates the fault detection process in the `FaultDetector`.

We measure the failure detection time as the time between the failure of the primary replica and the time when the `FaultDetector` actually detects a failure. To measure the failure detection time, we recorded two time stamps:

1. The first time stamp was recorded when the primary was killed.
2. The second timestamp was recorded when the `FaultDetector` detects the failure of the primary.

Analysis of failure detection time. Figure 7 shows the variation of detection times of the SEMI-ACTIVE replication over several iterations. Figure 8 shows the minimum, average, and maximum failure detection times for different polling intervals under WARM_PASSIVE replication style.

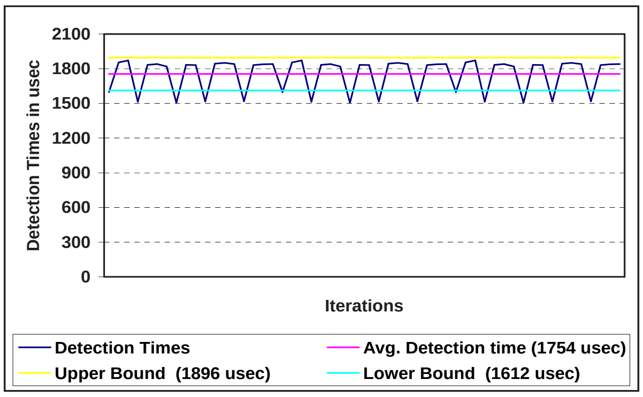

Figure 7: **Average and Bounds on Failure Detection Time within the Object Group for SEMI-ACTIVE Replication**

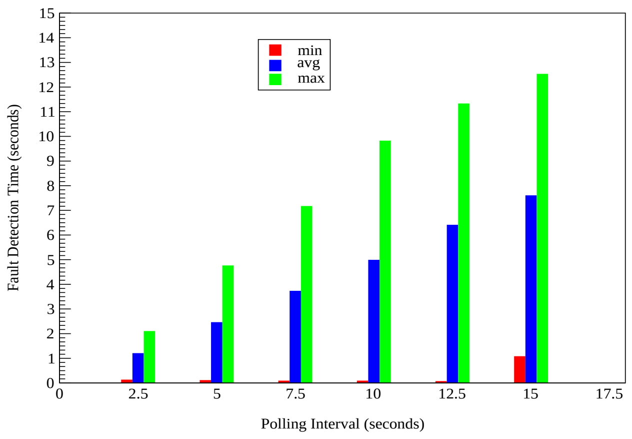

Figure 8: **Effect of Different Polling Intervals on Failure Detection Times for WARM_PASSIVE Replication**

The results in the above figures indicate the following:

- Failure detection is in the millisecond range for the SEMI-ACTIVE, and for the WARM_PASSIVE style is dependent on the polling interval.
- The bounds are in a narrow range and approximately ±4-5% of the average in SEMI-ACTIVE and for the WARM_PASSIVE is around ± 100%.

The results from WARM_PASSIVE illustrate that polling intervals should be minimized for applications requiring fast failover. But overly small polling intervals increase the number of messages in the network, however, which may be problematic over low-speed network links. The bounded nature of the detection times exhibited by the SEMI-ACTIVE replication styles are characteristics that DRE systems require.

5.1.2 Comparing Fault Detection and Recovery Times on the Client

Rationale. A client invoking a remote operation will experience some delay if its server fails during the operation. This delay has three parts:

1. The time taken by the infrastructure to detect the fault
2. The time taken by the infrastructure to promote a backup to become the primary, and
3. The time taken by the client to detect a failed primary and make invocations on the secondary.

Below, we describe the experiment conducted to compare the combination of these times for the SEMI-ACTIVE and the WARM_PASSIVE style. This time actually indicates the bounds on latency that the client will experience when faults occur.

Methodology for SEMI-ACTIVE. The experimental setup is similar to the one described in Section 5.1.1. To measure the effect of failures—and to compute the total recovery time—we allow the client to shutdown the primary by invoking the server object's `shutdown()` operation.

We measure the failure detection time as the time interval between when the client invoked the `shutdown()` operation to the time when the client can make the next request to the secondary. The time interval includes the error handling and the time needed for the client to retrieve an established connection and make the request to the secondary.

Methodology for WARM_PASSIVE. As described in Section 5.1.1, to measure the effect of failures, and to compute the total recovery time, we allow clients to connect to the primary replica. After clients connect to the primary replica, we terminate the primary replica by invoking the server object's `shutdown()` operation. For the client failover measurement, we start a timer in the client when the `shutdown()` operation is invoked.

When the DOORS's `FaultDetector` detects a failure it reports this failure to the `ReplicationManager`. In turn, the `ReplicationManager` selects a backup copy amongst the replicas and promotes it to become the new primary. Simultaneously, the `ReplicationManager` creates a new backup to maintain a consistent replica group size. It then notifies the new primary of its change in status by invoking the primary's `become_primary()` operation, which enables the new primary to respond to client requests. At this point, we stop our client timer and compute the failover time observed by the client.

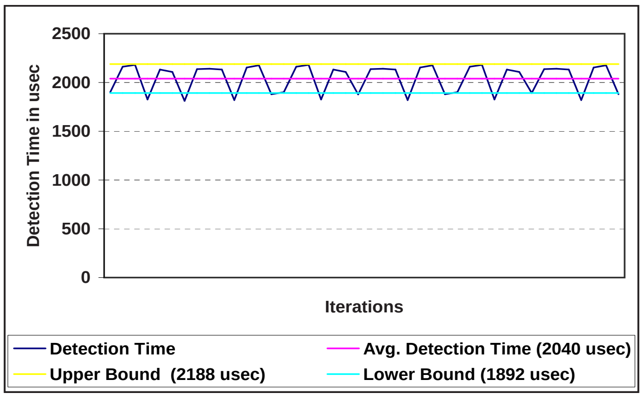

Figure 9: **Average and Bounds on Recovery Time for the Client in SEMI-ACTIVE Replication for a Stateless CORBA Server**

Analysis of failure detection time. Figure 9 shows the variation of detection times over several iterations for the SEMI-ACTIVE. The recovery times measured for the WARM_PASSIVE benchmark are shown in Figure 10. Figures 9 and 10 shows behavior similar to the one described in section 5.1.1. Bounded and predictable detection times on the client offered by the SEMI-ACTIVE style suggest that this style is better suited to DRE systems than the WARM_PASSIVE style.

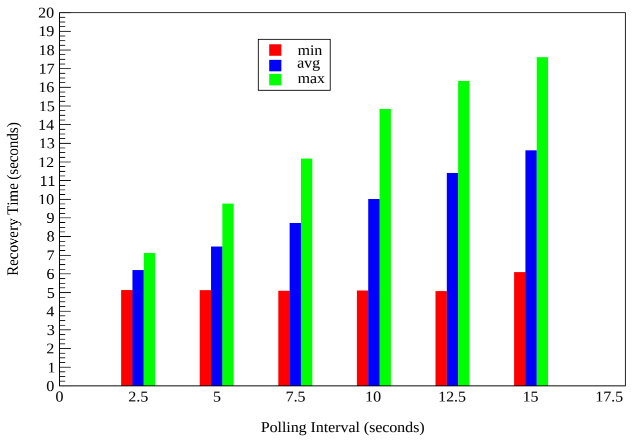

Figure 10: **Effect of Different Polling Intervals on Recovery Times for WARM_PASSIVE Replication for a Stateless CORBA Server**

5.2 Comparing Semi-Active Replication with Active Replication

This section describes the results of the empirical benchmarking studies we conducted to measure how well the SEMI-

ACTIVE replication style compares to the ACTIVE replication style in providing real-time and fault-tolerance support to DRE systems. The goal of this benchmark is to show the predictability in the *synchronization time* required to synchronize the state during every invocation. The infrastructure is similar to the one explained in Section 5.1.

Rationale. A client invoking a remote operation will experience additional delay if the server multicasts the requests or multicasts the state updates reliably to all the replicas, in addition to executing the invocation on the primary. Below, we describe the experiment conducted to measure the combined time, which is the actual delay experienced by a client for every invocation.

This experiment measures the latency experienced by the client when making invocations on an object group possessing state synchronization capabilities using the SEMI-ACTIVE replication style. We assume at this point that the ACTIVE replication without any extra protocols is not predictable, as shown in [35].

Methodology. Rather than modeling a communication subsystem that makes invocations to all the secondaries, we used TAO's Real-time Event Channel [9] to propagate state information to all the replicas with every invocation. We chose TAO's Real-time Event Channel for the following reasons:

1. The Event Channel offers a "push-pull" communication model, where all the registered event suppliers can publish events of interest to registered consumers.
2. TAO's Event Channel has been used in many production DRE systems, including real-time avionics mission computing [44], distributed interactive simulation [45], and large-scale network management [46].

The primary in the SEMI-ACTIVE replication style acts as a supplier of events to the channel and all the replicas subscribe to the channel as consumers to receive events. To add reliability to the delivery of events to the channel through the `push()` operation, we use the SYNC_WITH_SERVER reliable one-way policy at the ORB level.

We measured the time the client takes to make every invocation on the remote object. We varied the number of replicas receiving state information and captured the minimum, maximum, and average times. Eight bytes of data was transferred from the primary to the secondary. We also calculated the upper and lower bounds associated with this.

Analysis of latencies from state transfer. Figure 11 shows the variation of minimum, average, and maximum latency associated with communicating with primaries with varying number of replicas in the configuration outlined above. Figure 12 shows the average, upper, and lower bounds on the latency in the same experiments. The results in these figures indicate the following:

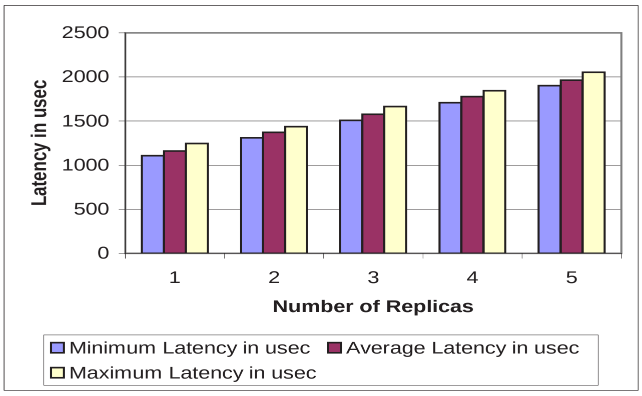

Figure 11: **Latency on the Client with Increase in Number of Replicas for SEMI-ACTIVE REPLICATION**

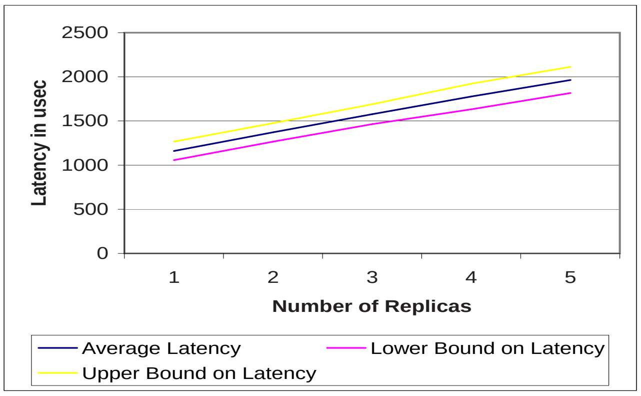

Figure 12: **Bounds on Latency with Increase in Number of Replicas for SEMI-ACTIVE Replication**

1. The latencies increase with the number of replicas observed by the client and
2. The small variance of latencies is being maintained with an increase in the number of replicas.

The increase in latency occurs since we use reliable oneways as opposed to a regular CORBA oneway call. Adding reliability to message transmission to every replica incurs additional overhead, as indicated by the results. A key engineering challenge, therefore, is striking the right balance between the degree of replication and the affordable latency reduction that developers of DRE systems can afford.

The above experiment indicates that the state transfer times remain tightly bounded for SEMI-ACTIVE with increase in the number of replicas. As the SEMI-ACTIVE replication style does not possess other known sources of unpredictable similar to ACTIVE style, we conclude that the SEMI-ACTIVE is better suited for DRE systems.

5.3 Summary of Results and Recommendations

Based on the results presented above, we now describe some of the key challenges that the SEMI-ACTIVE replication style resolves when designing dependable DRE systems. Though the empirical evaluation is made in the context of dependable DRE systems, this strategy is applicable to a larger class of distributed applications requiring dependability support from DOC middleware.

Challenge 1. A non-trivial amount of time is spent by the FT middleware to detect and recover from faults, which is antithetical to the latency and performance requirements of DRE systems.

Resolution. The SEMI-ACTIVE replication style that we propose places less overhead on the infrastructure to detect and recover from faults. Similar to the distributed computing paradigm which distributes computation between different nodes, the SEMI-ACTIVE replication style distributes the overhead of fault detection and recovery between replicas. The detection and recovery times from the SEMI-ACTIVE replication style are bounded allowing the possibility of accomplishing critical real-time tasks even in the presence of faults.

Challenge 2. Synchronizing message processing order across all replicas deterministically using the ACTIVE replication style calls for the usage of protocols with high overhead.

Resolution. The SEMI-ACTIVE replication imposes the primary's order of message execution on all the replicas in the group. The primary decides on the message order based on the local real-time QoS parameters and imposes that on all the replicas. The primary could either choose to multicast the request chosen for processing to all the replicas or choose to multicast the state information at the end of request processing. This flexibility is particularly useful for a class of DRE systems where the state of the system changes with external events in addition to CORBA requests.

Challenge 3. The ACTIVE replication style in order to maintain replica consistency could constrain the way in which application objects interact with their environment, like preventing execution threads from doing tasks that could change object states not associated with CORBA requests.

Resolution. The SEMI-ACTIVE replication resolves this challenge by allowing the primary to send state updates to all its secondaries in addition to CORBA requests from the clients. This gives applications the necessary flexibility to schedule and dispatch tasks that could affect the state of the object. Finally, we present an observation and a recommendation based on our empirical evaluation of the SEMI-ACTIVE replication style earlier in this section.

Observation. Client latencies tend to increase as a function of an increasing degree of replication.

Recommendation. Middleware researchers and implementors should carefully study application use cases, failure rates of DRE systems, and expected performance from the system to determine the replication degree automatically. Being able to configure the degree of replication adaptively would help simplify the development and deployment of DRE systems. Likewise, DRE system developers should carefully evaluate the trade-offs associated with increased replication degrees on the performance of their systems.

6 Related Work

CORBA is increasingly being adopted as the middleware of choice for mission-critical, large-scale, and heterogeneous DRE systems. The need to develop dependable middleware to support DRE systems has therefore motivated research on policies and mechanisms for fault-tolerant CORBA. As outlined in Section 1, research efforts devoted to enhancing the fault tolerance of CORBA ORBs and/or CORBA applications can be categorized into the following three strategies:

1. The integration strategy. This strategy layers the ORB on top of a reliable multicast communication subsystem. The ORB is modified to provide the necessary fault tolerance support, which will likely make the ORB non-compliant with the CORBA standard. The extent of the ORB modifications depends on the functionality that is being added, *e.g.*, a reliable, totally ordered group communication mechanism, such as Totem [33], could be added to deliver CORBA requests. For instance, a modified ORB can be linked with the client application as shown in Figure 13, and used to convert the ORB

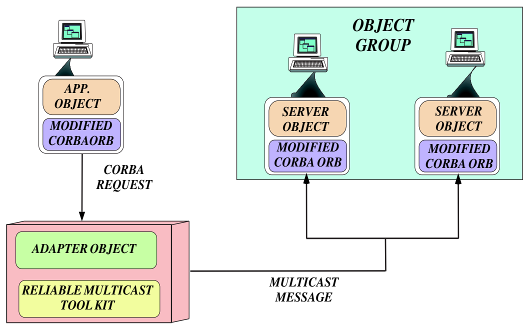

Figure 13: **Integration Fault Tolerance Strategy**

requests into multi-cast messages of the underlying toolkit.

Examples of the integration strategy include Orbix+Isis [47, 11] and Electra [47, 12]. Orbix+Isis was the first commercially available system that supported fault-tolerant CORBA-compliant applications. The fault-tolerance of server implementations is achieved using active (hot) replication, with reliable multicast support from low-level group communication system [11]. Electra was another early implementation of reliable CORBA. It differed from Orbix+Isis mainly with respect to its ease of use and its adaptability to other communication subsystems, such as Horus and Ensemble. The integration strategy requires modifications to ORB interfaces and language mappings, however, which reduces the portability of server and client applications.

2. The interception strategy. In this strategy, requests made by client objects are captured *externally* to the ORB, *e.g.*, via an OS-level interceptor [13], such as the `/proc` file system in UNIX. As shown in Figure 14, the interceptor can modify

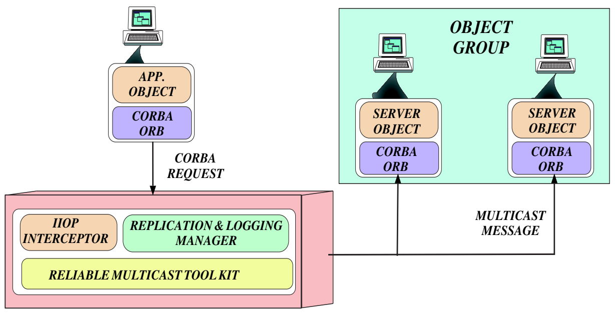

Figure 14: **Interception Fault Tolerance Strategy**

the client request parameters to alter the behavior of the application or to enhance the application with new functionality. The modified requests are then mapped onto a reliable group communication messaging system.

The interception strategy is used heavily in the Eternal system [10, 48] to construct a reliable ORB environment. Eternal intercepts system calls made by the client object through the lower-level I/O subsystem and maps these system calls to the reliable multicast subsystem. Eternal differs from Orbix+Isis or Electra in that Eternal does not modify the ORB or the language mapping. It therefore ensures the transparency of fault-tolerance to applications. The AQuA framework [15] also provides a variant of the interception approach. The AQuA gateway acts as an intermediary between the CORBA objects and the Ensemble group communication subsystem and translates GIOP messages to group communication primitives. In addition, AQuA uses the Quality Objects (QuO) [49] framework to allow applications to specify their reliability/availability requirements.

The Maestro toolkit [34] developed at Cornell University, uses a combination of interception and integration approaches. It provides an object-oriented wrapper facade [13] to process-based group communication systems and supports transparent client-side failover through the use of multi-profile object references. The process group exports a multi-profile IOR to the client and the client uses a single profile in that IOR to connect to the primary of the group through an IIOP gateway.

3. The service strategy. In this strategy, a set of interfaces and objects are defined as a CORBA service that provides the policies and mechanisms for delivering fault tolerance to applications. Fault tolerance can therefore be provided as a part of the standard suite of CORBA Services, without requiring extensive modifications to CORBA ORBs. Figure 15 illustrates this strategy. An *object group* is registered with the fault

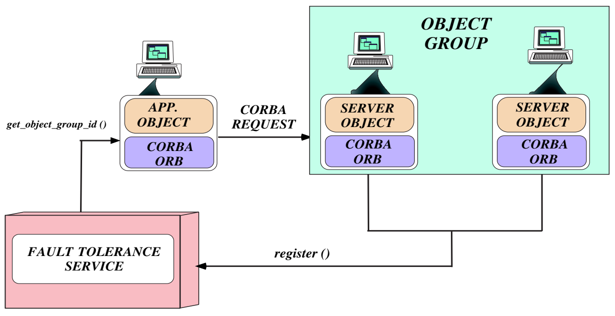

Figure 15: **Service Fault Tolerance Strategy**

tolerant service. Client applications query the service to obtain the object group and make invocations on the object group. The service manages the replicas and the state of their associated objects.

The service strategy is used in the Object Group Service (OGS) [19]. OGS provides a group of services, including membership service, monitoring service, consensus service and messaging service, that work together to enhance the reliability of an application through both active and passive replication. The DOORS [16] framework described in this paper is another example of a service-based approach to fault tolerance.

Our recent work on DOORS [17, 18] and TAO's pluggable protocols framework [28] has focused on a service strategy that also allows application developers to select the protocol(s) used to communicate between collaborating ORB endsystems. The CORBA specification defines a standard transport protocol, IIOP, which runs atop TCP. There are many use-cases, however, particularly in telecommunication and real-time systems, where ORBs must communicate over non-TCP protocols, such as VME, ATM, SCTP, and shared memory. Moreover, researchers have devised other messaging protocols, such as HTTP-NG [50] or GIOP-lite [28], that have a smaller "wire footprint" than the standard CORBA GIOP messaging protocol.

The three types of research strategies outlined above have yielded incompatible fault tolerant solutions to the CORBA model that are not portable or interoperable. To address this problem, therefore, the OMG devised the Fault-tolerant CORBA (FT-CORBA) specification [1], which leveraged the lessons learned from earlier R&D efforts [20, 21, 22, 19, 10, 16]. As we pointed out in Sections 1 and 3, however, FT-CORBA is not yet usable in DRE systems that possess stringent end-to-end QoS constraints. The SEMI-ACTIVE approach described in this paper is a step towards adding this important capability to CORBA.

The SEMI-ACTIVE approach described here has been heavily influenced by the work on Delta-4 (XPA) architecture [36] and the Fault-Tolerance work on the Distributed Ada-95 project at EPFL [51]. A variation of the passive replication style called as SEMI-PASSIVE replication was used by EPFL [52] to improve the recovery time after crash. Though this methodology can provide predictable detection and recovery times, this methodology makes the same assumptions pointed out in Section 3.1 which makes it unsuitable for DRE systems. More particularly, the SEMI-PASSIVE style doesn't address how state changes associated with real-time events other than remote invocations can be propagated to the replicas. Further, the SEMI-PASSIVE replication style is hard to implement without changing the interfaces and semantics of FT-CORBA. Our work focusses on the SEMI-ACTIVE approach to support simultaneous dependability and timeliness requirements, while complying with the FT-CORBA interfaces and semantics.

Other related work [37, 38, 53] presents principles and mechanisms to achieve fault-tolerance in real-time systems via Time-triggered Message-triggered Objects [54], which have been explored at the University of California, Irvine. The work that has been presented in this paper is in the context of CORBA and RT-CORBA and we expect this work to resolve some open R&D challenges [55].

7 Concluding Remarks and Future Directions

Distributed real-time and embedded (DRE) systems are playing an increasingly important role in many application domains, including telecommunication networks (*e.g.*, high-speed central office switching), telemedicine (*e.g.*, remote surgery), manufacturing process automation (*e.g.*, hot rolling mills), and aerospace (*e.g.*, avionics mission computing). Although there are many types of DRE systems, they have one thing in common: *the right answer delivered too late becomes the wrong answer*. Providing the right answer at the right time is therefore imperative for mission-critical DRE systems.

Our effort at adding dependability to DRE systems focuses on developing and deploying standard CORBA middleware that can provide timeliness and performance guarantees to applications even when crash faults occur. Our future plan is to collect more empirical data and demonstrate that SEMI-ACTIVE replication style provides timeliness and performance guarantees by amortizing the unpredictability in detection and recovery over successful invocations. Our goal in this effort is to provide *the right answer at the right time* by lowering the infrastructure overhead needed to detect and recover from faults. *The ACE ORB (TAO), its FT-CORBA infrastructure, and the benchmarks described in this paper are available as open-source software from `deuce.doc.wustl.edu/Download.html`.*

While our work presented in this paper helps address the problems presented in Sections 3.1 and 3.2 above, the problems of Sections 3.3 and 3.4 remain open challenges to the DOC middleware R&D community. We conclude by outlining future research directions for the DOC middleware community to address the challenges outlined in Sections 3.3 and 3.4.

Byzantine and partial failures. The detection, diagnosis, and response to failures other than crash failures is an exceedingly hard problem to address in an application-independent manner. For example, consider an object that is returning correct responses too slowly. The fault may be in the component, in the connection with the component, or in some otherwise unrelated component that is sharing some resource with the slow component. These issues have received intense and protracted study [56]. Perhaps the most promising direction for the middleware fault-tolerant community would be to provide standard interfaces through which application-specific detection, diagnosis, and response mechanisms can act.

Comparable QoS benchmarks. The ability to compare qualities of service across different infrastructure implementations requires an agreement on the set of qualities that apply to each service, and rigorous definitions of those qualities. For example, if "invocation latency" is agreed to be a relevant quality of a client-server service, then it must be determined exactly how this value will be measured. Since any set of measurements of such qualities will present a distribution of values, the useful statistics that are to be derived from such a distribution must be agreed on, *e.g.*, worst-case or average and standard deviation. It would also be helpful to have standard benchmarking suites for these qualities.

Although solving the problems presented above is clearly hard, the consequences of solving them would be profound: it would be possible to construct DRE systems that reliably meet their functional and quality requirements, and do so when operating on any sufficiently powerful infrastructure.

References

[1] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 3.0 edition, June 2002.

[2] Michi Henning and Steve Vinoski, *Advanced CORBA Programming with C++*, Addison-Wesley, Reading, MA, 1999.

[3] Object Management Group, *Real-Time CORBA Specification*, 1.1 edition, Aug. 2002.

[4] Douglas C. Schmidt and Fred Kuhns, "An Overview of the Real-time CORBA Specification," *IEEE Computer Magazine, Special Issue on Object-oriented Real-time Computing*, vol. 33, no. 6, June 2000.

[5] Douglas C. Schmidt and et al., "TAO: A Pattern-Oriented Object Request Broker for Distributed Real-time and Embedded Systems," *IEEE Distributed Systems Online*, vol. 3, no. 2, Feb. 2002.

[6] Rebecca Callison, Marilynn Goo, and Daniel Butler, "Real-time CORBA Trade Study," Tech. Rep. D204-31159, The Boeing Company, 1999.

[7] David Corman, "WSOA-Weapon Systems Open Architecture Demonstration-Using Emerging Open System Architecture Standards to Enable Innovative Techniques for Time Critical Target (TCT) Prosecution," in *Proceedings of the 20th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Oct. 2001.

[8] Carlos O'Ryan, Douglas C. Schmidt, Fred Kuhns, Marina Spivak, Jeff Parsons, Irfan Pyarali, and David Levine, "Evaluating Policies and Mechanisms for Supporting Embedded, Real-Time Applications with CORBA 3.0," in *Proceedings of the 6^{th} IEEE Real-Time Technology and Applications Symposium*, Washington DC, May 2000, IEEE.

[9] Timothy H. Harrison, David L. Levine, and Douglas C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," in *Proceedings of OOPSLA '97*, Atlanta, GA, Oct. 1997, ACM, pp. 184–199.

[10] Priya Narasimhan, *Transparent Fault Tolerance for CORBA*, Ph.D. thesis, University of California, Dept. Of Electrical and Computer Engineering, Santa Barbara, CA, Dec. 1999, Available as Technical Report UCSB 99-18.

[11] Kenneth Birman and Robbert van Renesse, *Reliable Distributed Computing with the Isis Toolkit*, IEEE Computer Society Press, Los Alamitos, 1994.

[12] Silvano Maffeis, "Adding Group Communication and Fault-Tolerance to CORBA," in *Proceedings of the Conference on Object-Oriented Technologies*, Monterey, CA, June 1995, USENIX.

[13] Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*, Wiley & Sons, New York, 2000.

[14] Priya Narasimhan, Louise E. Moser, and P. M. Melliar-Smith, "Using Interceptors to Enhance CORBA," *IEEE Computer*, vol. 32, no. 7, pp. 64–68, July 1999.

[15] M. Cukier, J. Ren, C. Sabnis, W.H. Sanders, D.E. Bakken, M.E. Berman, D.A. Karr, and R.E. Schantz, "AQuA: An Adaptive Architecture that provides Dependable Distributed Objects," in *IEEE Symposium on Reliable and Distributed Systems (SRDS)*, West Lafayette, IN, Oct. 1998, pp. 245–253.

[16] P.Y. Chung, Y. Huang, S. Yajnik, D. Liang, and C.Y. Shih, "Providing Fault Tolerance to CORBA Applications," in *Poster at Middleware '98*, Lake District, England, Sept. 1998.

[17] Balachandran Natarajan, Aniruddha Gokhale, Douglas C. Schmidt, and Shalini Yajnik, "DOORS: Towards High-performance Fault-Tolerant CORBA," in *Proceedings of the 2^{nd} International Symposium on Distributed Objects and Applications (DOA 2000)*, Antwerp, Belgium, Sept. 2000, OMG.

[18] Balachandran Natarajan, Aniruddha Gokhale, Douglas C. Schmidt, and Shalini Yajnik, "Applying Patterns to Improve the Performance of Fault-Tolerant CORBA," in *Proceedings of the 7^{th} International Conference on High Performance Computing (HiPC 2000)*, Bangalore, India, Dec. 2000, ACM/IEEE.

[19] Pascal Felber, Rachid Guerraoui, and André Schiper, "The Implementation of a CORBA Object Group Service," *Theory and Practice of Object Systems (TAPOS)*, vol. 4, no. 2, pp. 93–105, Feb. 1998.

[20] Paul D. Ezhilchelvan, Raimundo A. Macedo, and Santosh K. Shrivastava, "Newtop: A Fault-Tolerant Group Communication Protocol," in *International Conference on Distributed Computing Systems*, 1995, pp. 296–306.

[21] C. Marchetti, A. Virgillito, and R. Baldoni, "Design of an Interoperable FT-CORBA compilant Infrastructure ," in *In the Proceedings of the 6th International Workshop on Object Oriented Real-time Dependable Systems (WORDS'01)*, Rome, Italy, Jan. 2001.

[22] Jean-Charles Fabre and Tanguy Perennou, "Friends - a flexible architecture for implementing fault tolerant and secure distributed applications," in *European Dependable Computing Conference*, 1996, pp. 3–20.

[23] Joseph K. Cross and Douglas C. Schmidt, "Applying the Quality Connector Pattern to Optimize Distributed Real-time and Embedded Middleware," in *Patterns and Skeletons for Distributed and Parallel Computing*, Fethi Rabhi and Sergei Gorlatch, Eds. Springer Verlag, 2002.

[24] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1995.

[25] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee, "The Design and Performance of Real-Time Object Request Brokers," *Computer Communications*, vol. 21, no. 4, pp. 294–324, Apr. 1998.

[26] Christopher D. Gill, David L. Levine, and Douglas C. Schmidt, "The Design and Performance of a Real-Time CORBA Scheduling Service," *Real-Time Systems, The International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware*, vol. 20, no. 2, Mar. 2001.

[27] Fred Kuhns, Douglas C. Schmidt, and David L. Levine, "The Design and Performance of a Real-time I/O Subsystem," in *Proceedings of the 5^{th} IEEE Real-Time Technology and Applications Symposium*, Vancouver, British Columbia, Canada, June 1999, IEEE, pp. 154–163.

[28] Carlos O'Ryan, Fred Kuhns, Douglas C. Schmidt, Ossama Othman, and Jeff Parsons, "The Design and Performance of a Pluggable Protocols Framework for Real-time Distributed Object Computing Middleware," in *Proceedings of the Middleware 2000 Conference*. ACM/IFIP, Apr. 2000.

[29] Douglas C. Schmidt, Sumedh Mungee, Sergio Flores-Gaitan, and Aniruddha Gokhale, "Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers," *Journal of Real-time Systems, special issue on Real-time Computing in the Age of the Web and the Internet*, vol. 21, no. 2, 2001.

[30] Alexander B. Arulanthu, Carlos O'Ryan, Douglas C. Schmidt, Michael Kircher, and Jeff Parsons, "The Design and Performance of a Scalable ORB Architecture for CORBA Asynchronous Messaging," in *Proceedings of the Middleware 2000 Conference*. ACM/IFIP, Apr. 2000.

[31] Aniruddha Gokhale and Douglas C. Schmidt, "Measuring the Performance of Communication Middleware on High-Speed Networks," in *Proceedings of SIGCOMM '96*, Stanford, CA, Aug. 1996, ACM, pp. 306–317.

[32] Irfan Pyarali, Carlos O'Ryan, Douglas C. Schmidt, Nanbor Wang, Vishal Kachroo, and Aniruddha Gokhale, "Applying Optimization Patterns to the Design of Real-time ORBs," in *Proceedings of the 5^{th} Conference on Object-Oriented Technologies and Systems*, San Diego, CA, May 1999, USENIX.

[33] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos, "Totem:A Fault-Tolerant Multicast Group Communication System," *Communications of ACM* , 1996.

[34] Alexey Vaysburd and Ken Birman, "The Maestro approach to Building Reliable Interoperable Distributed Applications with Multiple Execution Styles," *Theory and Practice of Object Systems (TAPOS)*, vol. 4, no. 2, pp. 73–80, Feb. 1998.

[35] S. Poledna, A. Burns, A.J. Wellings, and P. Barrett, "Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems," *IEEE Transactions on Computers*, vol. 49, no. 2, pp. 100–111, 2000.

[36] P. Barrett, A. Hilborne, P. Bond, D. Seaton, P. Verissimo, L. Rodrigues, and N. Speirs, "The Delta-4 Extra Performance Architecture (XPA)," in *Proceedings of the 20th Int. Symp. on Fault-Tolerant Computing Systems (FTCS-20)*, 1990.

[37] Kane Kim and Subbaraman C, "PSRR: A Scheme for Time-Bounded Fault Tolerance in Distributed Object-Based Systems," in *Proc. IEEE High-Assurance Systems Engineering (HASE) Workshop*, Ontario, Canada, Oct. 1996, IEEE.

[38] Kane Kim and Subbaraman C, "Fault-Tolerant Real-Time Objects," *Communications of the ACM*, Jan. 1997.

[39] L. Rodrigues, A. Casimiro, and P. Verissimo, "Priority-based Totally Ordered Multicast," in *Proceedings of the 3rd IFAC/IFIP workshop on Algorithms and Architectures for Real-Time Control (AARTC'95).* , May 1995.

[40] Douglas C. Schmidt and Stephen D. Huston, *C++ Network Programming, Volume 1: Mastering Complexity with ACE and Patterns*, Addison-Wesley, Boston, 2002.

[41] Douglas C. Schmidt and Stephen D. Huston, *C++ Network Programming, Volume 2: Systematic Reuse with ACE and Frameworks*, Addison-Wesley, Reading, Massachusetts, 2002.

[42] Gautam Thaker and Patrick Lardieri and Chuck Winters, "SCTP and its Adaptation to TAO ," in *Proceedings of the 2^{nd} Annual TAO Workshop*, July 2002.

[43] VITA Standards Organization, "Myrinet-on-VME Protocol Specification Draft Standard," http://www.myri.com/open-specs, 2000.

[44] David C. Sharp, "Reducing Avionics Software Cost Through Component Based Product Line Development," in *Proceedings of the 10th Annual Software Technology Conference*, Apr. 1998.

[45] Carlos O'Ryan, Douglas C. Schmidt, and J. Russell Noseworthy, "Patterns and Performance of a CORBA Event Service for Large-scale Distributed Interactive Simulations," *International Journal of Computer Systems Science and Engineering*, vol. 17, no. 2, Mar. 2002.

[46] Guru Parulkar, Douglas C. Schmidt, Eileen Kraemer, Jon Turner, and Anshul Kantawala, "An Architecture for Monitoring, Visualization, and Control and Gigabit Networks," *IEEE Network*, vol. 11, no. 5, September/October 1997.

[47] Sean Landis and Silvano Maffeis, "Building Reliable Distributed Systems with CORBA," *Theory and Practice of Object Systems*, vol. 3, no. 1, pp. 31–43, 1997.

[48] Louise Moser, P. Melliar-Smith, and Priya Narasimhan, "A Fault Tolerance Framework for CORBA," in *International Symposium on Fault Tolerant Computing*, Madison, WI, June 1999, pp. 150–157.

[49] John A. Zinky, David E. Bakken, and Richard Schantz, "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems*, vol. 3, no. 1, pp. 1–20, 1997.

[50] W3C HTTP-NG Working Group, "W3C HTTP-NG Protocol," www.w3.org/Protocols/HTTP-NG, 1998.

[51] Thomas Wolf and Alfred Strohmeier, "Fault tolerance by transparent replication for distributed ada 95," in *International Conference on Reliable Software Technologies - Ada-Europe'99, Santander, Spain, June 7-11 1999*, Michael Gonzalez Harbour and Juan A. de la Puente, Eds. 1999, number 1622, pp. 412–424, Springer-Verlag.

[52] Xavier Defago, Andre Schiper, and Nicole Sergent, "Semi-passive replication," in *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*. IEEE, 1998, pp. 43–50.

[53] Kane Kim and C. Subbaraman, "Principles of Constructing a Timeliness-Guaranteed Kernel and Time-triggered Message-triggered Object Support Mechanisms," in *Proceedings of the International Symposium on Object-Oriented Real-time Distributed Computing (ISORC)*. IEEE/IFIP, Apr. 1998.

[54] Kane Kim, "APIs Enabling High-Level Real-Time Distributed Object Programming," *IEEE Computer Magazine, Special Issue on Object-oriented Real-time Computing*, June 2000.

[55] Kane Kim, "Issues Insufficiently Resolved in Century 20 in the Fault-Tolerant Distributed Computing Field," in *Proceedings of Symposium on Reliable Distributed Systems)*. IEEE, 2000.

[56] Miguel Castro and Barbara Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, Feb. 1999.