DOORS: Towards High-performance Fault Tolerant CORBA

Balachandran Natarajan
Dept. of Computer Science
Washington University
St. Louis, MO 63130
bala@cs.wustl.edu

Aniruddha Gokhale, Shalini Yajnik
Bell Laboratories
Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ 07974
{agokhale, shalini}@lucent.com

Douglas C. Schmidt
Dept. of Electrical and Computer Engineering
University of California
Irvine, CA 92697
schmidt@uci.edu

This paper appeared in the Proceedings of the 2^{nd} Distributed Applications and Objects (DOA) conference, Antwerp, Belgium, Sept. 21-23, 2000.

Abstract

An increasing number of applications are being developed using distributed object computing middleware, such as CORBA. Many of these applications require the underlying middleware, operating systems, and networks to provide end-to-end quality of service (QoS) support to enhance their efficiency, predictability, scalability, and fault tolerance. The Object Management Group (OMG), which standardizes CORBA, has addressed many of these application requirements recently in the Real-time CORBA and Fault-tolerant CORBA specifications.

This paper provides four contributions to the study of fault-tolerant CORBA middleware for performance-sensitive applications. First, we provide an overview of the Fault Tolerant CORBA specification. Second, we describe a framework called DOORS, which is implemented as a CORBA service to provide end-to-end application-level fault tolerance. Third, we outline how the DOORS' reliability and fault-tolerance model has been incorporated into the standard OMG Fault-tolerant CORBA specification. Finally, we outline the requirements for CORBA ORB core and higher-level services to support the Fault Tolerant CORBA specification efficiently.

Keywords: Fault-tolerant CORBA, Middleware protocols.

1. Introduction

Emerging trends: Developers of distributed applications are increasing using the standard services and protocols defined by distributed object computing middleware, such as the Common Object Request Broker Architecture (CORBA) [1]. CORBA is a distributed object computing middleware standard defined by the Object Management Group (OMG) that allows clients to invoke operations on remote objects without concern for where the object resides or what language the object is written in [2]. In addition, CORBA shields applications from non-portable details related to the OS/hardware platform they run on and the communication protocols and networks used to interconnect distributed objects. These features make CORBA ideally suited to provide the core communication infrastructure for distributed systems and applications.

A growing number of next-generation applications demand varying degrees and forms of quality of service (QoS) support from their middleware, including efficiency, predictability, scalability, and dependability. In CORBA-based middleware, this QoS support is provided by Object Request Broker (ORB) endsystems [3]. ORB endsystems consist of network interfaces, operating system I/O subsystems, CORBA ORBs, and higher-level CORBA services.

Our prior research on CORBA middleware has explored the efficiency, predictability, and scalability aspects of ORB endsystem design, including static and dynamic scheduling, event processing, I/O subsystem and pluggable protocol integration, synchronous and asynchronous ORB Core architectures, systematic benchmarking of multiple ORBs, and optimization principle patterns for ORB performance. This paper focuses on two other dimensions in the ORB endsystem design space: *(1) identifying key aspects of CORBA implementations that support both high-performance and fault tolerant properties and (2) demarcating the responsibility between the service layer and ORB layer to support fault tolerance.*

Limitations with current fault tolerance strategies: There is a large body of research literature [4] and tools that focus on improving the reliability and recoverability of application *processes*. For instance, Firstwatch [5], Watchd [6], and Wolfpack [7] are tools designed to improve the fault tolerance of application processes. Techniques focusing solely on process-based failure detection and recovery, however, are not necessarily applicable to CORBA applications for the following reasons:

- **Overly coarse granularity:** Each application process may contain several simultaneously active CORBA objects and threads. If individual objects or threads fail, *e.g.*, due to a crash or abnormal termination, a process may not terminate, however. Thus, a fault tolerance strategy that detects only process failures may not identify these finer-grained types of problems.

- **Inability to restore complex object relationships:** Distributed objects communicate with other distributed objects via object references. When a failed object is recovered, all object references for other local or remote CORBA objects held just before failure must be recovered. Likewise, other CORBA objects holding object references to the failed CORBA object must obtain new references, as well. Moreover, object references can be persistent *i.e.*, the references can be reassigned to new servants if the original has shutdown gracefully or abnormally. In contrast, processes do not possess these characteristics. Thus, restoring object reference relationships is a complex problem that is unique to distributed object computing middleware and is not handled well by process-based recovery strategies.

- **Restrictions on process checkpointing and recovery:** Distributed objects often maintain state that must be checkpointed periodically to survive crash failures. Fault tolerance strategies used for process checkpointing may incur excessive overhead or may be unable to checkpoint the desired state for all objects. Thus, a finer level of granularity for checkpointing process state is needed to permit individual objects in a process to checkpoint their internal state independently of each other.

As a result of the limitations with process-based reliability and recovery outlined above, it is necessary to explore new strategies or adapt existing strategies to provide high reliability and availability to CORBA-based distributed object computing applications.

Solution approach → Supporting fault tolerance for CORBA objects: Fault tolerance strategies for distributed object computing middleware, such as CORBA, address the limitations of process-based fault tolerance. In general, research on fault tolerance for CORBA ORBs and its applications can be divided into the three strategies [8], an *integration* strategy, an *interception* strategy, and a *service* strategy, which are outlined below:

1. Integration strategy: In this strategy, 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 could be added to deliver CORBA requests. For instance, a modified ORB can be linked with the client application as shown in Figure 1, and used to convert the ORB re-

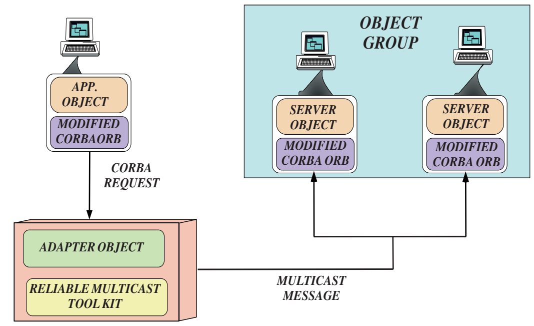

Figure 1. Integration Fault Tolerance Strategy

quests into multi-cast messages of the underlying toolkit. The *adapter object* linked with the ORB provides the mapping to convert ORB requests to multi-cast messages. Orbix+Isis [4] and Electra [9] are examples of the integration strategy.

2. Interception strategy: In this strategy, requests made by client objects are captured *externally* to the ORB, *e.g.*, via an OS-level interceptor [10], such as the `/proc` file system in UNIX. As shown in Figure 2, the interceptor can modify 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 on to a reliable group communication messaging system. The Eternal system [11], and the AQuA framework [12] are examples of the interception strategy.

3. 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. Thus, fault tolerance can be provided as a part of the standard suite of CORBA Services, without requiring extensive modifications to CORBA ORBs. Figure 3

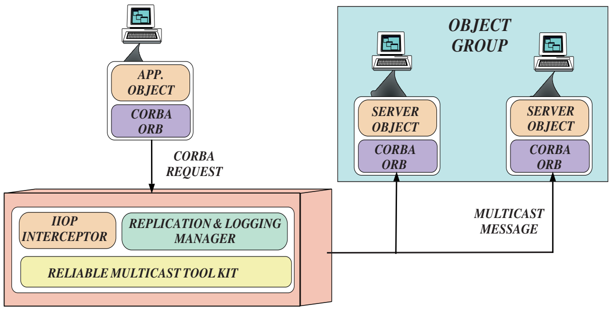

Figure 2. Interception Fault Tolerance Strategy

illustrates this strategy. An *object group* is registered with

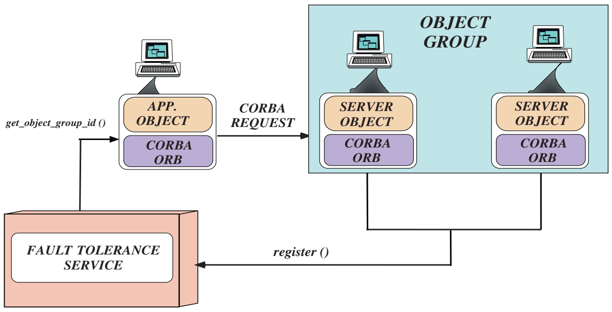

Figure 3. Service Fault Tolerance Strategy

the fault 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 Distributed Object-Oriented Reliable Service (DOORS) [13], which is our focus in this paper, is an example of this strategy.

The *interception* and *service* strategies outlined above have been incorporated in the OMG Fault Tolerant CORBA (FT-CORBA) specification [14], which was adopted as a standard in January, 2000.

Paper organization: The remainder of this paper is organized as follows: Section 2 summarizes the recently adopted Fault Tolerant CORBA (FT-CORBA) specification, illustrates the concepts from DOORS that were incorporated into this standard, and outlines the design of DOORS; Section 3 presents the key ORB-level and service-level features that an FT-CORBA implementation must support to simultaneously achieve high reliability *and* performance; and Section 4 presents concluding remarks.

2. Overview of Fault Tolerant CORBA

The Fault Tolerant CORBA (FT-CORBA) [14] specification defines a standard set of interfaces, policies, and services that provide robust support for applications requiring

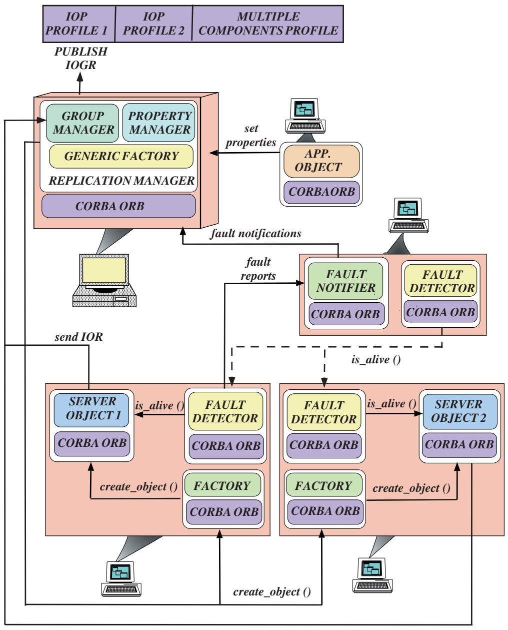

Figure 4. The Architecture of Fault Tolerant CORBA

high reliability. The fault tolerance mechanism used to detect and recover from failures is based on *entity redundancy*. Naturally, in FT-CORBA the redundant entities are replicated CORBA objects.

This section presents an overview of the FT-CORBA standard and explains how this standard is being realized with the Distributed Object-Oriented Reliable Service (DOORS). DOORS is a prototype implementation of FT-CORBA that implements a core subset of the functionality defined by the standard. Through our contributions and active participation in the OMG working group on Fault Tolerant CORBA, the concepts pioneered by DOORS and key lessons learned in its development have been integrated into the FT-CORBA standard.

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" [15] composite object. This strategy allows greater flexibility in configuration management of the replicas.

Figure 4 illustrates the key components in the FT-CORBA architecture. All components shown in the figure 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.

Interoperable object group references (IOGRs): FT-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 uniquely [1]. In addition, it defines an IOR for composite objects called the *interoperable object group reference* (IOGR), which is illustrated in Figure 5. The

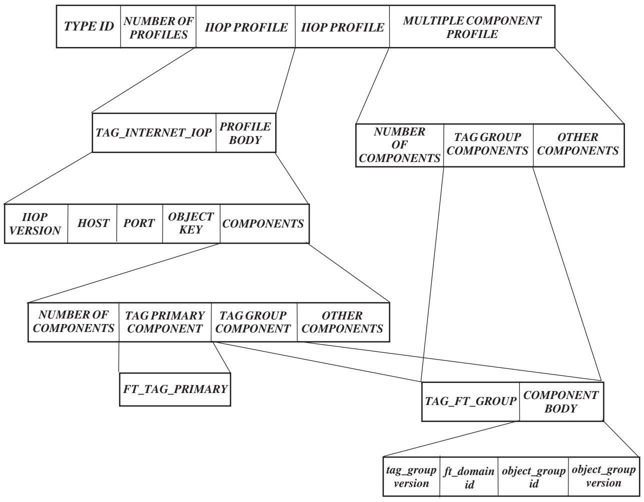

Figure 5. Example of an IOGR

IOGR contains multiple TAG_INTERNET_IOP profiles any of which may 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. Clients 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 cycles through the object references contained in the IOGR until the request is handled successfully by a replica object. The references in the IOGR are considered invalid only if all server objects fail, in which case an exception is propagated to the client application.

Replication manager: This component is responsible for managing replicas and contains the following three components:

1. PropertyManager: This component 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. Example replication styles include the following:

- COLD_PASSIVE – In this replication style, the replica group contains a single primary replica that responds to client messages. If a primary fails an idle replica is spawned on-demand to function as the new primary.
- WARM_PASSIVE – In the WARM_PASSIVE replication style, the replica group contains a single primary replica that responds to client messages. In addition, one or more backup replicas are pre-spawned to handle crash failures. If a primary fails a backup replica is selected to function as the new primary and a new backup is created to maintain a constant replica group size.
- ACTIVE – In the ACTIVE replication style all replicas are primary and handle client requests independently of each other. To ensure a single reply sent to the client and to maintain consistent state amongst the replicas, a special group communication protocol is necessary.

Membership of a group and data consistency of the group members can be controlled either by the FT-CORBA infrastructure or by applications. FT-CORBA standardizes both application-controlled and infrastructure-controlled membership and consistency styles.

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

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

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 periodically polls applications to determine if their objects are "alive." FT-CORBA requires application objects to implement a `PullMonitorable` interface that exports an `is_alive` operation. A *push-based* monitoring mechanism can also be implemented. In this scheme, which is also known as a "heartbeat monitor," applications implement a `PushMonitorable` interface and send periodic heartbeats to the `FaultDetector`.

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

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.

Logging and recovery: For the application-controlled consistency style, applications are responsible for their own failure recovery. For the infrastructure-controlled consistency style, however, FT-CORBA defines a logging and recovery mechanism. This mechanism is responsible for intercepting and logging CORBA GIOP messages from client objects to servers. Figure 6 illustrates how the logging

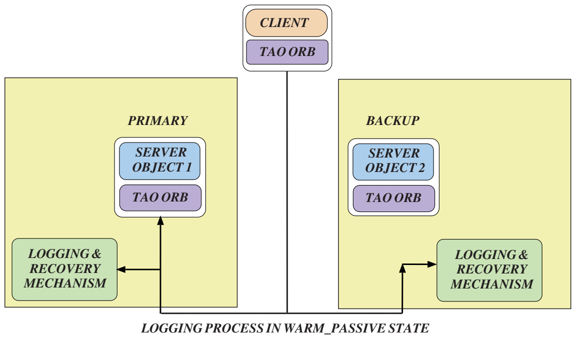

Figure 6. Operation of the Logging Mechanism

mechanism operates during normal operation. Once a fault occurs, as a part of the recovery action, the messages that are recorded are played back to the new primary. This will get the new primary in a state consistent with the old primary before failure. Figure 7 illustrates how the recovery

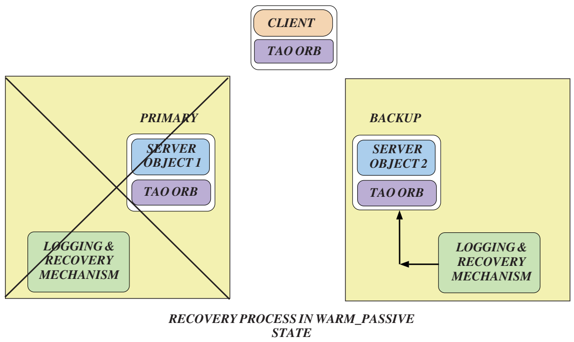

Figure 7. Operation of the Recovery Mechanism

mechanism applies messages from the log on to the replica to get it to the current state. The backup member in a object group with WARM_PASSIVE replication style should receive state updates at constant intervals of time during normal operation. During recovery, when a backup member is promoted to a primary, the recovery mechanism applies to the backup member only the recent state updates after the last complete update. For an object group configured with the COLD_PASSIVE replication style, a new backup is created and the recovery mechanism applies the complete log to the backup.

After all replicas are consistent, the recovery mechanism then re-invokes the operations that were made by the client, but which did not execute due to the primary replica's failure. In addition, it retrieves a consistent state for the new replica. The logging and recovery mechanism ensures that failovers are transparent to applications.

Fault Tolerance Domains: To manage very large and complex applications, the FT-CORBA standard defines a feature called *fault tolerance domains*. The purpose of fault tolerance domains is 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 8. In addition, hosts can be part of several domains

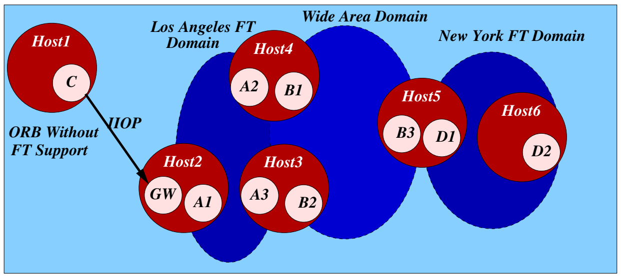

Figure 8. Fault Tolerant CORBA Domains

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`. The FT-CORBA standard allows both intra- and inter-domain invocations of requests on object groups.

2.2. Requirements of the FT-CORBA Standard

The FT-CORBA standard imposes the following requirements on CORBA 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: The standard defines three new *tagged components* into the object reference model. These tagged components are used 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 object computing system. As a result, each component described above must itself be replicated and a mechanism provided to deal with potential failures and recovery, as described in Section 2.4.

Bounded fault detection and notification: The FT-CORBA standard mandates a timely detection and notification of faults to the `ReplicationManager` and subsequent recovery from failures.

Transparent failovers: Recovery from failure and managing the size of the replica object group is transparent to clients making requests to that object group. Moreover, as mentioned earlier, client applications should not need to distinguish between making requests to a replicated object or a non-replicated object.

Transparent client redirection and reinvocation: As explained earlier, the standard defines an IOGR which the client ORB uses to send requests to the replica object group. 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 is transparent to the client application.

To maintain the *at-most-once* semantics of the CORBA object model, the FT-CORBA standard defines a REQUEST service context that the client ORB includes in requests. The REQUEST service context includes a unique *client ID* for the client, a *retention ID*, and an *expiration time* for the request. 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 the GROUP_VERSION service context that the client ORB sends in a request. This service context includes the group version number of the replica group to which it sends a request. The server ORB uses this information to determine if the 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.

2.3. Limitations of the FT-CORBA Standard

Due to the diverse set of fault-tolerance requirements and the large variety of distributed applications requiring fault-tolerance, the current version of the FT-CORBA standard compromises on the number of interfaces, policies, and features it provides. As a result, FT-CORBA vendors are free to provide proprietary extensions. The objective in specifying the standard is to gain insight into the types of extensions required to satisfy additional demands imposed by a range of applications. These insights are intended to drive subsequent revisions of the standard.

The limitations imposed by the current FT-CORBA standard include the following:

Legacy ORBs: A client hosted by a legacy ORB not supporting the FT-CORBA standard can continue to invoke operations on the replicated server. However, it will not benefit from the fault-tolerant properties offered by the replicated server.

Vendor dependences: All hosts belonging to a single fault tolerance domain must use ORBs from the same vendor to ensure interoperability and a higher degree of fault tolerance beyond what is specified by the FT-CORBA standard. This limitation stems from the fact that the vendors are free to provide proprietary extensions to overcome lack of provision of certain domain-specific interfaces or policies in the FT-CORBA standard.

Deterministic behavior: For the infrastructure-controlled consistency style, deterministic behavior is required of application objects, as well as ORBs, to ensure strong replica consistency.

Inability to handle certain types of faults: The FT-CORBA standard provides no mechanism to handle faults due to network partitioning. In addition, it cannot handle (1) *commission faults*, where incorrect results are produced by the hosts or objects and (2) *correlated faults*, where errors are caused due to design or programming faults.

2.4. Impact of DOORS on the FT-CORBA Standard

DOORS was developed prior to the FT-CORBA standard as an experimental fault tolerant CORBA middleware. Through our contributions and active participation in the OMG working group on Fault Tolerant CORBA, the concepts pioneered by DOORS and the lessons learned implementing fault tolerant CORBA middleware have been integrated into the FT-CORBA standard. The current version of DOORS implements a subset of the functionality of the FT-CORBA standard. Below, we describe the FT-CORBA functionality offered by DOORS and present the interaction protocol among the DOORS components. This description also outlines how the DOORS model has influenced the FT-CORBA standard.

Figure 9 illustrates the interaction of key components in the DOORS framework. As shown in the figure, DOORS has a `ReplicationManager` component that

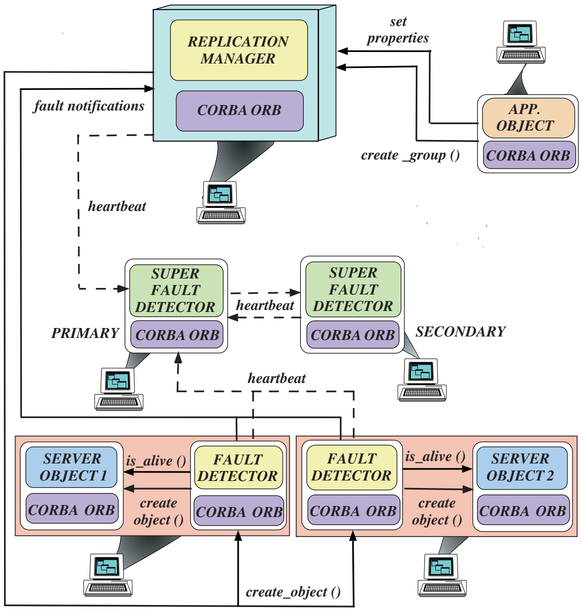

Figure 9. Components in the DOORS Fault Tolerance Architecture

encapsulates most of the property management and group management functionality defined in FT-CORBA. The `FaultDetector` and the `Super FaultDetector` components support hierarchical fault detection and notification. The `FaultDetectors` perform object-level fault detection and `Super FaultDetectors` perform host-level fault detection. By making the `FaultDetector` and `Super FaultDetector` CORBA objects, they can detect the liveness of CORBA objects and perform the same functions as process-level monitors.

In addition, `FaultDetectors` and `Super FaultDetectors` act as `FaultNotifiers` and propagate fault reports to the `ReplicationManager`. `FaultDetectors` support both push-based (heartbeats) and pull-based (polling) object monitoring. All FT-CORBA infrastructure components, *i.e.*, the `FaultDetector` and `ReplicationManager`, are monitored by a `Super FaultDetector` via heartbeats. Thus, DOORS uses replication and monitoring of infrastructure components to ensure there is no single point of failure in the system.

DOORS uses a service strategy to provide fault tolerance to CORBA applications. This strategy has been incorporated into the FT-CORBA standard. The service-level management components of FT-CORBA are derived directly from DOORS' service-level management components, such as its `ReplicationManagers` and `FaultDetectors`.

2.5. Overview of the FT-CORBA/DOORS Fault Tolerance Protocol

Figure 10 illustrates the protocol interactions between the components of the DOORS framework when an appli-

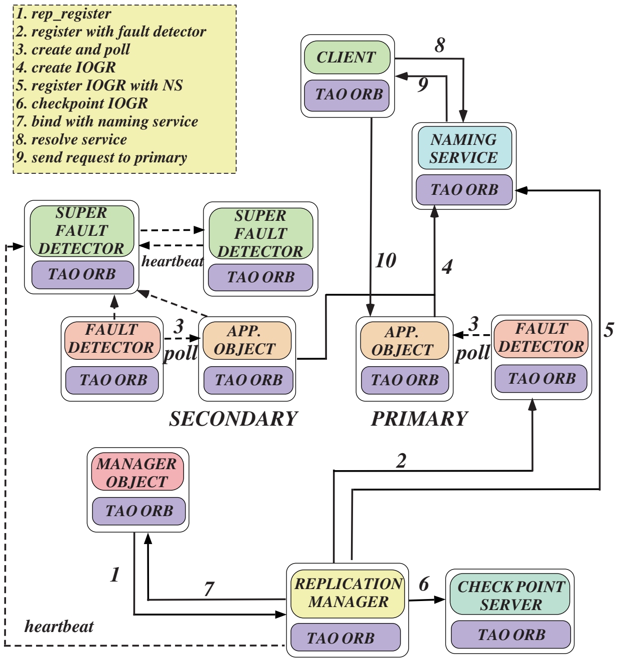

Figure 10. FT-CORBA Component Interaction Protocol in DOORS

cation uses a WARM_PASSIVE scheme. As shown in the figure, establishing a replica group using this scheme involves the following steps:

1. An application manager can request the `ReplicationManager` to create a replica group using the `create_object` operation of the FT-CORBA's `GenericFactory` interface and passing it a set of fault tolerance properties for the replica group.
2. The `ReplicationManager`, as mandated by the FT-CORBA standard, delegates the task of creating individual replicas to local factory objects based on the `ObjectLocation` property. The local factories return individual object references of created objects to the `ReplicationManager`.
3. At this point, the `ReplicationManager` informs `FaultDetectors` to start monitoring the replicas.
4. The `ReplicationManager` collects all the IORs of the individual replicas, creates an IOGR for the group, and designates one of the replicas as a primary.

For an `active` replication style, all replicas will be designated as primaries.

5. The `ReplicationManager` then registers the IOGR with the Naming Service, which publishes it to other CORBA applications and services.
6. The `ReplicationManager` checkpoints the IOGR and other state.
7. A client interested in the service contacts the Naming Service.
8. The Naming Service responds with the IOGR.
9. Finally, the client makes a request and the client ORB ensures that the request is sent to the primary replica.

Depending on the monitoring style chosen, *e.g.*, heartbeat vs. polling, the `FaultDetector` will continue monitoring the replicas at periodic intervals after a replica group is established. `FaultDetectors` and `ReplicationManager` send heartbeats to a primary "`Super`" `FaultDetector` at periodic intervals. Since FT-CORBA does not allow single points of failures the `Super FaultDetector` itself is replicated. These replicas send heartbeats to each other at periodic intervals. One such `Super FaultDetector` is designated as a primary and others are backups. If the primary fails, the other backups elect a new primary.

2.6. Unimplemented FT-CORBA features of DOORS

This section describes features of the FT-CORBA standard not implemented in the current version of DOORS.

Active replication style: Although the DOORS' `ReplicationManager` is programmed to handle object group creation and recovery that uses the ACTIVE replication style, there is no provision in the current version to use a group communication protocol. We are working on an implementation that uses TAO's pluggable protocols framework [16], which will enable DOORS to be configured with any group communication protocol that can be configured into the framework.

Infrastructure-controlled logging and recovery: The current version of DOORS just supports application-controlled logging and recovery mechanisms. We are integrating a solution based on *portable interceptors* to support infrastructure-controlled logging and recovery.

Fault notifier: The current DOORS implementation does not provide a `FaultNotifier`. Instead, the fault detection and notification functionality is combined in the `FaultDetectors`.

2.7. DOORS Extensions to the FT-CORBA standard

This section describes the extensions DOORS provides to address several limitations of the FT-CORBA standard or lack of features from certain ORB.

Enhanced IOGR: For each group of n replicas in DOORS, the `ReplicationManager` creates an interoperable object group reference (IOGR) with $n + 1$ profiles. The last profile contains a reference to the `ReplicationManager`, with the object group's identifier encapsulated inside the profile.

When an IOGR is published to the client, the client can invoke operations via the IOGR. When an object fails to respond to polling from the `FaultDetectors`, or if the `FaultDetectors` fail to receive heartbeats from the replicas, the `FaultDetectors` signal a failure to the `ReplicationManager`. The `ReplicationManager` starts a new backup as a corrective action. At that instant, the IOGR held by the client becomes a "stale" object reference since the constituents of the groups have changed. The client continues to use the same IOGR, however, since the other constituents in the IOGR are still valid.

After all replicas in the IOGR have been tried, the client ORB uses the last reference to make a call on the `ReplicationManager`. The `ReplicationManager` which is aware of the new IOGR for the object group uses the GIOP LOCATE_FORWARD mechanism to forward the new IOGR to the client ORB. The client ORB thereafter uses the new IOGR to invoke requests on the object group. This scheme is useful when the underlying ORB does not provide support for the GROUP_VERSION service context that detects stale IOGRs.

Fault notification: The DOORS `FaultDetectors` use proprietary interfaces to notify faults to the `ReplicationManager`. The DOORS fault notification interfaces enable the `ReplicationManager` to locate the failed object group in its internal tables. In contrast, the interfaces provided by FT-CORBA are less efficient because they are based on the interfaces defined by the CORBA *Notification* service, which use CORBA `Anys` to pass information.

Additional properties: DOORS defines additional properties to identify the replica type, *e.g.*, PRIMARY, BACKUP, or IDLE. In addition, DOORS defines a property that determines how many missed heartbeats to allow before declaring a replica object dead. This property is important to prevent unnecessary fault detection and recovery, which is expensive and degrades system availability until recovery is completed.

3. Requirements for High-performance, Fault Tolerant CORBA

Our experience implementing the FT-CORBA specification and the TAO ORB [3] has provided many insights on the requirements for implementing high-performance and fault tolerant middleware. Below, we describe these requirements, which we categorize into (1) ORB Core-specific and (2) FT-CORBA service-specific.

3.1. ORB Core Requirements for High-performance FT-CORBA

This section describes the requirements that must be addressed to implement a high-performance ORB Core to support FT-CORBA.

IOGR parsing and connection establishment: To send a request, the FT-CORBA standard recommends that the client ORB first establish a connection to the TAG_FT_PRIMARY replica component of the TAG_INTERNET_IOP profile. Since the FT-CORBA standard does not mandate the location of the TAG_FT_PRIMARY component, the ORB core must provide an efficient strategy of parsing the IOGR to extract the desired component.

Implementing efficient IOGR component parsing is complicated for the passive replication style when the primary replica has failed and an alternate replica is promoted to the primary. In this case, the client ORB has a stale IOGR. Thus, any request made to the primary replica component in the stale IOGR will fail and the client ORB must reissue the request on alternate replicas belonging to the server object group.

If the alternate replica to which a connection is established and request is reissued is not a primary, the client ORB will receive one or more LOCATION_FORWARD messages, thereby degrading response time. Thus, the ORB Core should provide efficient and predictable IOGR parsing and handling of LOCATION_FORWARD messages.

As explained in Section 2.2, the server ORB is required to parse both incoming client requests and service their contexts. If it receives a GROUP_VERSION service context, the server ORB must determine based on the group version number if the IOGR in the client request is stale. In this event, the server ORB must first respond with an LOCATE_FORWARD_PERM exception to the client and pass it the latest IOGR. The server ORB should implement these steps efficiently to improve response time and increase availability.

2. Reliable handling of messages: As mentioned in Section 2.2, in the event of failure, the client ORB must retry the request to other components of the IOGR. Below, we outline three scenarios that make this task challenging to implement efficiently.

- **Client receives no reply:** Several factors can cause a client not to receive a reply. For example, a server object may have failed, a tw-way request may execute for an extended time, or network congestion may delay a reply. Normally, the client ORB or communication protocol associates a timeout with every invocation to the server. If the delay in receiving a reply is longer than the timeout, the client ORB does not propagate an exception to the application. Instead, it pings the server to determine liveness.

  If the server does not respond to the ping, the client ORB attempts to contact the server over alternative network paths specified in the IOGR. If that fails too, then the client ORB tries to establish a connection with an alternate replica. A poor implementation of this scheme could result in jitter and reduce predictability, which is unacceptable for certain mission-critical applications.

- **Failure of server object processing:** In an infrastructure-controlled consistency style, the server ORB is required to track the CORBA requests relative to the target object group. When a client request arrives at the server ORB, it can be in two different states if the target object crashes:

1. The target object could have failed just before an upcall is made to the servant.
2. The target object could have failed after an upcall to the servant is made.

The information logged by the server ORB's logging mechanism should cater to these different failure conditions. To guarantee predictable response time, therefore, efficient implementations of the logging and recovery mechanism must be provided.

- **Duplicate client messages:** To handle duplicate client messages and preserve CORBA's "at-most-once" semantics, the server ORB must parse the incoming client request to extract the REQUEST service context. This information identifies the request that the server ORB must use to consult the log and determine if it is a duplicate. Efficient implementations of request parsing for the service contexts and log lookups are essential to guarantee bounded and predictable response time to clients.

3.2 Service-level requirements for Fault Tolerant CORBA

For FT-CORBA to be widely used, it is imperative that the FT-CORBA service layer incur negligible impact on performance. This section describes the FT-CORBA service requirements that must be addressed to provide high performance.

Support for dynamic system configuration: The FT-CORBA standard provides the interfaces to dynamically set fault tolerance properties, such as the fault monitoring interval or minimum number of replicas. Dynamic configuration is essential for fine tuning the system for high performance, *e.g.*, for the `PULL`-based monitoring style the average failure detection time is half the polling interval. Thus, for a large polling interval, the failure detection time will be large and unacceptable for systems requiring high availability. In contrast, decreasing the polling interval to a small value increases the overhead of end-to-end messages, thereby degrading performance.

Bounded recovery time: Applications using passive replication styles will incur a transient period during failure recovery when the system is unavailable. Thus, for applications requiring high availability, the time to recover from failures should bounded. This in turn can help improve performance since response time is more predictable, thereby reducing the number of unnecessary retransmissions. For active replication styles, the overhead to maintain replica consistency should be negligible.

Minimal overhead of the FT-CORBA components: Since the FT-CORBA standard does not allow single points of failure, fault-tolerance must be provided for the components of the FT-CORBA service, such as `Fault Detectors` and the `Replication Manager`. Fault detection and recovery of these components should add negligible overhead to the system.

4. Concluding Remarks

A growing number of CORBA applications with stringent performance requirements also require fault tolerance support. The most flexible strategy for providing fault tolerance to CORBA applications is via higher-level CORBA services integrated with an enhanced ORB Core. To address these necessary enhancements in a standard manner, the OMG recently adopted the Fault Tolerant CORBA (FT-CORBA) specification [14].

To make FT-CORBA usable by performance-sensitive applications it must incur negligible overhead. To address these requirements, therefore, an FT-CORBA implementation should possess the following characteristics:

1. The fault detection and failovers incurred by servers should be transparent to clients.
2. Response time to the client should be efficient and predictable, irrespective of server failovers.
3. The overhead incurred by the FT-CORBA implementation should maintain application performance requirements, such as efficiency and scalability, within designated bounds end-to-end.

To achieve these goals, implementations of FT-CORBA must address the ORB Core and the service layer requirements described in this paper. We are currently applying a variety of architectural, design, and optimization principle patterns [17] to improve the performance of the DOORS implementation of FT-CORBA.

References

[1] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.3 edition, June 1999.

[2] Steve Vinoski, “CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments,” *IEEE Communications Magazine*, vol. 14, no. 2, February 1997.

[3] 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.

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

[5] Veritas, “Veritas FirstWatch,” www.veritas.com/us/products/firstwatch, 2000.

[6] Yennun Huang and Chandra Kintala, “Software Implemented Fault Tolerance: Technologies and Experience,” in *23rd International Symposium on Fault-tolerance Computing (FTCS)*, Toulouse, France, June 1993, pp. 2–10.

[7] MSCS, “Microsoft NT Server Edition,” www.microsoft.com, 1998.

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] Object Management Group, *Fault Tolerant CORBA Specification*, OMG Document orbos/99-12-08 edition, December 1999.

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

[16] 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.

[17] 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.