Gateways for Accessing Fault Tolerance Domains*

P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith

Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
priya@alpha.ece.ucsb.edu {moser,pmms}@ece.ucsb.edu

Abstract. Enterprise applications can be structured as domains, where each domain contains objects that are replicated for fault tolerance, with the replication being managed by a fault tolerance infrastructure local to the domain. Gateways can allow unreplicated clients to benefit from the fault tolerance services of the replicated servers, without compromising replica consistency within the fault tolerance domain. For CORBA-based enterprise applications, the gateway mechanisms can be implemented transparently to the ORB and to the application using interception; specific enhancements to existing ORBs make it possible for unreplicated clients to enjoy a higher degree of reliability.

1 Introduction

Applications are increasingly spanning enterprises across the Internet, with the application objects within one enterprise communicating with, and performing operations on, the application objects of another enterprise. The reliability of the application as a whole depends on the reliability of the objects in each of the communicating enterprises, which are separated possibly by a considerable distance, as shown in Fig 1. Each enterprise is likely to be, and indeed should be, responsible only for the reliability of the objects under its control, but each enterprise must nevertheless allow the objects of a different enterprise to communicate with its own objects without compromising the consistency of the replicated objects of either enterprise. The domain of control of the fault tolerance infrastructure of each enterprise constitutes a *fault tolerance domain*; different fault tolerance domains can be connected through a *gateway*.

The concepts of fault tolerance domains and gateways are not restricted to communication between enterprises. Internet-based applications such as stock trading involve customers using Web browsers (typically unreplicated thin clients) to communicate with the servers (typically replicated for fault tolerance) of a stock trading company. The unreplicated Web browser should not

* Research supported by the Defense Advanced Research Projects Agency in conjunction with the Office of Naval Research and the Air Force Research Laboratory, Rome, under Contracts N00174-95-K-0083 and F3602-97-1-0248, respectively.

J. Sventek and G. Coulson (Eds.): Middleware 2000, LNCS 1795, pp. 88–103, 2000.

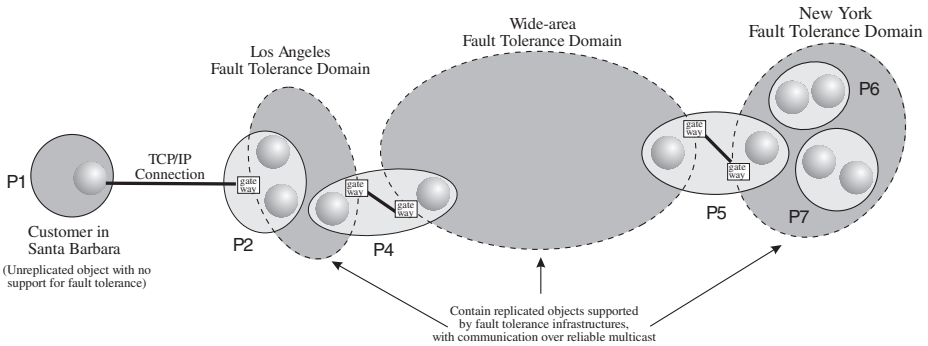

Fig. 1. Gateways bridge fault tolerance domains, and allow objects in one fault tolerance domain to communicate with those in another. P_i represents a processor hosting some application objects

need to be aware of the replication of the stock trading servers, but can nevertheless benefit from the fault tolerance of the servers. The unreplicated clients (the Web browsers) can be made to communicate with the replicated servers (the stock trading servers) through a gateway that hides the replication of the servers. The replicated servers are managed by the fault tolerance infrastructure of the stock trading company, and the gateway serves as the "entry point" into the fault tolerance domain. The gateway is a crucial element because it must "understand" the reliability mechanisms inside the fault tolerance domain, as well as the unreliable semantics of the external client, and must bridge these different semantics and mechanisms, without compromising the reliability of the objects within the fault tolerance domain.

A different motivation for a fault tolerance domain is that an application might have a large number of objects that require replication, and it might not be a scalable or feasible solution for a single fault tolerance infrastructure to manage the replication of all of these objects. Instead, it would be preferable to decompose the application into smaller collections of objects, with each collection of objects being managed by a distinct fault tolerance infrastructure, and therefore constituting a fault tolerance domain.

Regardless of the motivation for a fault tolerance domain, the gateway mechanism is identical and essential. In this paper, a gateway mechanism is described in the context of applications developed using the Common Object Request Broker Architecture (CORBA) [9] distributed object standard established by the Object Management Group (OMG). While CORBA currently does not provide support for fault tolerance, efforts [8] are underway within the OMG to standardize interfaces for fault-tolerant CORBA.

A key issue in fault tolerance for CORBA will be the mechanisms to support interaction of non-fault-tolerant CORBA systems with fault-tolerant CORBA systems. The gateways described in this paper address the issues in the implementation of such mechanisms, the problems in building those mechanisms using

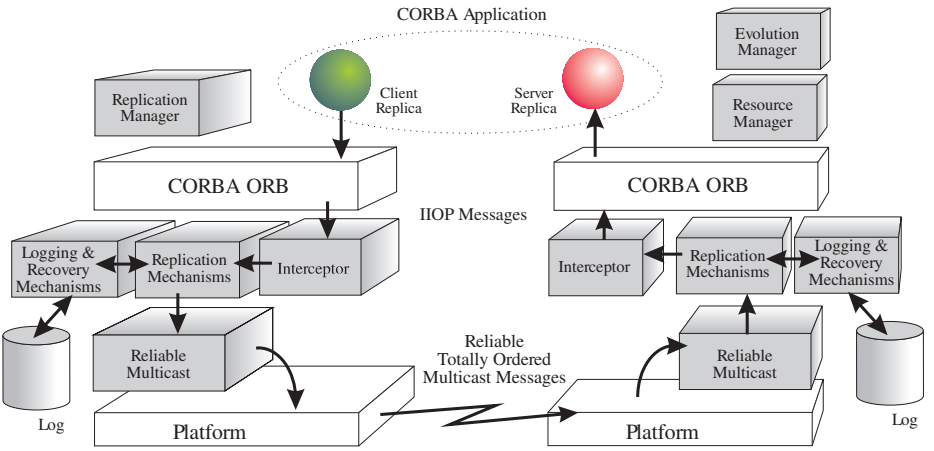

Fig. 2. The Eternal system – the fault tolerance infrastructure within the fault tolerance domain

existing ORBs, and enhancements to existing ORBs that might overcome these deficiencies.

2 The Fault Tolerance Infrastructure

The Eternal system [5,6] constitutes the fault tolerance infrastructure (within the fault tolerance domain) that provides reliability for applications running over commercial-off-the-shelf implementations of CORBA. The mechanisms implemented in different parts of the Eternal system work together efficiently to provide strong replica consistency with low overheads, and without requiring modification of either the application or the ORB.

In the Eternal system, the client and server objects of the CORBA application are replicated, and the replicas are distributed across the system. Active replication and passive replication of both client and server objects are supported. To facilitate replica consistency, the Eternal system conveys the Internet Inter-ORB Protocol (IIOP) messages of the CORBA application using an underlying reliable totally ordered multicast group communication system, such as Totem [4].

The structure of the Eternal system is shown in Figure 2. The Eternal Replication Manager replicates each application object, according to user-specified fault tolerance properties (including the choice of replication style – stateless, cold passive, warm passive, active, active with voting) and distributes the replicas across the system. The Eternal Resource Manager monitors the system resources, and maintains the initial and minimum number of replicas.

The Eternal Interceptor captures the IIOP messages (containing the client's requests and the server's replies), which are intended for TCP/IP, and diverts

them instead to the Eternal Replication Mechanisms for multicasting via Totem. The Eternal Replication Mechanisms, along with the Eternal Logging-Recovery Mechanisms, maintain the consistency of the replicas, detect and provide recovery from faults. The Replication Mechanisms and the Totem protocols run on every processor within a fault tolerance domain.

The Eternal Evolution Manager exploits object replication to support upgrades to the CORBA application objects. The Replication Manager, the Resource Manager and the Evolution Manager are themselves implemented as collections of CORBA objects and, thus, can themselves be replicated and thereby benefit from Eternal's fault tolerance capabilities. They need not be present on every processor; their replicas can run on any processor within the fault tolerance domain.

The technology of Eternal formed the basis of our response [2] to the Object Management Group's Request for Proposals [8] on fault-tolerant CORBA. With our close involvement in the ongoing OMG standardization process, it appears likely that the technology of Eternal will form the basis of the forthcoming OMG standard for fault tolerance for CORBA.

2.1 Transparency via Interception

The Eternal Interceptor [7] is a non-ORB-level, non-application-level component that transparently "attaches" itself to every executing CORBA object, without the object's or the ORB's knowledge, and is capable of modifying the object's behavior as desired. Because of its location underneath the ORB, Eternal's Interceptor is transparent to the ORB and to the application, and can be implemented in an ORB-independent manner.

Current operating systems provide "hooks", such as library interpositioning, that can be exploited to develop interceptors. Using library interpositioning, the Eternal Interceptor can transparently override the default definitions for the symbols in any dynamically linked library, without requiring modification of the ORB, the CORBA application or the operating system.

The library routines that an interceptor is made to redefine in a library-interpositioning implementation, depends on the extent of the information that the interceptor must extract (from the ORB or the CORBA application) to enhance the application with new features. The interceptor may capture all, or a particular subset, of the library routines used by the CORBA application, depending on the feature being added. The library interpositioning approach used by Eternal's Interceptor has no overheads in the path of message transmission, and can be deployed with various ORBs.

2.2 Strong Replica Consistency

The Replication Mechanisms, operating in concert with the Logging-Recovery Mechanisms, provide for strongly consistent replication of the CORBA application objects. Eternal provides support for the detection and suppression of duplicate invocations and duplicate responses, and for state transfer to new and

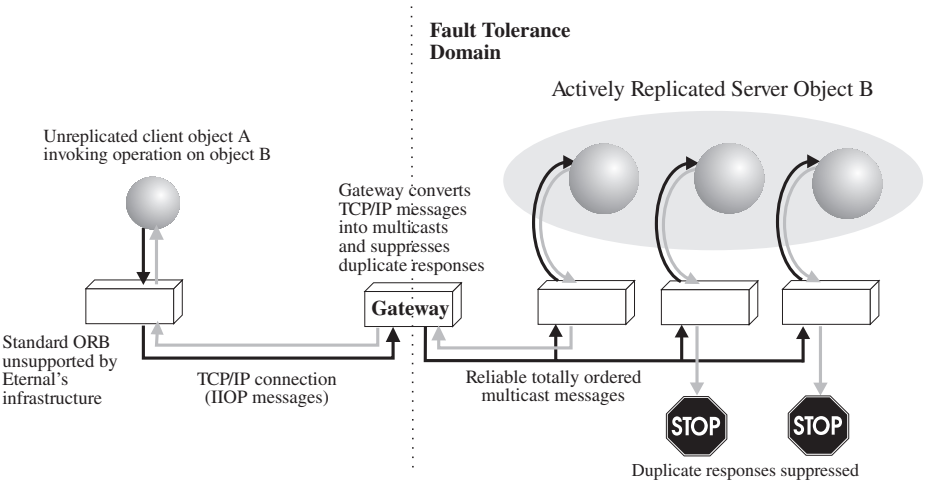

Fig. 3. Eternal's gateways allow unreplicated clients to communicate with replicated objects

recovering replicas for both actively and passively replicated objects. Most commercial applications and/or ORBs use multithreading, a significant source of non-determinism. To ensure strong replica consistency even for multithreaded CORBA objects that are replicated, Eternal employs special Interceptor-level mechanisms to enforce determinism for multithreaded CORBA applications, without requiring them to be modified.

While the fault tolerance infrastructure ensures strong replica consistency within the fault tolerance domain, it is the responsibility of the gateway to ensure that unreplicated clients wishing to contact replicated objects within the fault tolerance domain (through the gateway) do not compromise the replica consistency of those replicated objects.

3 Gateways to Fault Tolerance Domains

Eternal must allow the CORBA applications that it supports to communicate with unreplicated objects that are outside the fault tolerance domain, *i.e.*, Eternal's domain of control. Some of these unreplicated objects (*e.g.*, a Web browser on a personal computer that provides no fault tolerance) may not be supported by, or have access to, Eternal's fault tolerance infrastructure, and may run over standard IIOP-enabled ORBs.

Eternal ensures that these unreplicated objects outside the fault tolerance domain can nevertheless communicate with the replicated objects that are under Eternal's control inside the fault tolerance domain. Eternal makes this communication possible without the unreplicated object ever being aware of the existence of a fault tolerance domain, of the replication of the objects within

the fault tolerance domain, or of Eternal itself. Thus, Eternal extends the replication transparency that it provides to the application objects within the fault tolerance domain equally to unreplicated objects outside Eternal's control.

The gateways that the Eternal system provides serve as the "entry point" for unreplicated clients into the fault tolerance domain, and allow unreplicated external objects to invoke replicated Eternal-managed objects.

Within a fault tolerance domain:

- All objects are replicated, with the replication managed by Eternal's fault tolerance infrastructure. Each replicated object is assigned a unique object group identifier.
- Communication between replicated objects occurs through a reliable totally-ordered multicast protocol, thereby facilitating replica consistency, as described in Section 2.2. The Replication Mechanisms hosting the replicas of an object are addressed by multicasting messages to the object's group identifier.
- Replicated clients do not use the TCP/IP {host, port} information within the Interoperable Object Reference (IOR) of any of the server replicas to contact the replicated server. Instead, the Eternal Interceptor transparently diverts the socket establishment routines at every client replica to form a connection to the local Eternal Replication Mechanisms, which then multicast the notification of the connection establishment to the Replication Mechanisms hosting the server replicas.

Outside a fault tolerance domain:

- Objects are unreplicated, and are unaware of the internal mechanisms of, and the replication within, the fault tolerance domain
- Communication occurs through CORBA's TCP/IP-based Internet Inter-ORB Protocol (IIOP)
- Clients use the TCP/IP {host, port} information within the Interoperable Object Reference (IOR) of the target server to establish a connection with the server.

Unreplicated objects outside the fault tolerance domain must never be allowed to access the replicated objects within the fault tolerance domain directly. Such direct communication, if permitted, would violate replica consistency. The reason is that the unreplicated client can communicate only through TCP/IP, thereby implying that it would contact only *one* of the server replicas, and invoke an operation on that replica alone.

If the server is actively replicated, and only the single invoked server replica performs the operation, it may have a different state from that of the other replicas of the server object, resulting in inconsistent replication. If the server is passively replicated, and the single primary replica is invoked, the primary replica might itself invoke nested operations as a result of the original invocation. If the primary fails before it receives the results of the nested invocations, a new primary server replica will be elected. However, because the new primary

(formerly a backup replica) did not receive the original invocation, it will not be able to handle the returned responses from the nested invocations and to return a response to the original invocation. Thus, to ensure replica consistency, the replicas of an object are contacted through a reliable totally-ordered multicast, and not individually through TCP/IP.

Additional mechanisms are provided by Eternal so that an IOR published by a replicated object within the fault tolerance domain "point" the external clients in the direction of the IIOP-enabled gateway, rather than the target replicated object. However, the external client that uses this IOR is unaware of this. When using the information in the IOR for connection establishment, the client implicitly assumes that the endpoint is the real server and, thus, sends IIOP invocations (destined for the server) to the gateway.

Note that the gateway is not a CORBA object, but constitutes part of the mechanisms provided by the fault tolerance infrastructure of Eternal. However, by receiving the unreplicated client's IIOP invocations without returning exceptions, and by forwarding the replicated server's IIOP responses to the unreplicated clients, the gateway appears to the client to be a remote CORBA server object.

To perform the invocation (response) forwarding into (out of) the fault tolerance domain, the gateway must be able to interpret the IIOP messages sent over TCP/IP connections from outside the fault tolerance domain, as well as the reliable totally-ordered multicast protocol messages within the fault tolerance domain, and must provide the necessary translation between them. This functionality of the gateway is shown in Figure 3.

Another aspect of the gateway is that it must "hide" the replication of the servers from the external client. This involves detecting duplicate responses returned by the replicas of the server, and filtering out only a single distinct response to the external client. In addition, the gateway must itself be reliable so that it does not constitute a single point of failure.

3.1 Connection Establishment

When a gateway is used, every unreplicated external client must continue to "believe" that the remote endpoint to which it connects (using the information in the server IOR) is the server when, in fact, the remote endpoint is the gateway. This can be done by ensuring that the addressing information in the IOR is the {gateway_host, gateway_port} and that the gateway always returns the expected IIOP responses to the client's IIOP invocations so that the client never suspects otherwise.

Eternal replaces the {server_host, server_port} in the IOR of each server replica with the {gateway_host, gateway_port} through the use of its Interceptor. The intent of the Interceptor is to interpose at the point that the server-side ORB queries the operating system for the host and the port information, *prior* to publishing the IOR. By modifying the *getsockname()* call and/or the *sysinfo()* call (with the SI_HOSTNAME command) to return the gateway_host and the gateway_port instead of the server_host and the server_port, respectively, the IOR

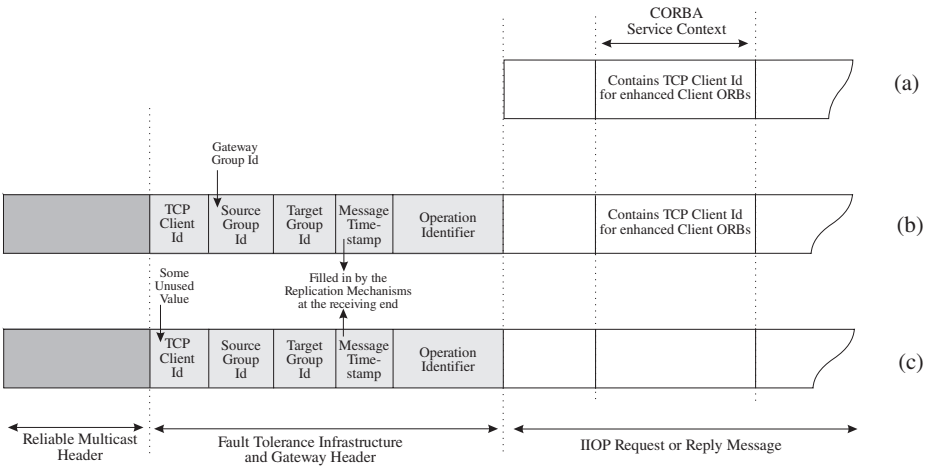

Fig. 4. Messages sent (a) between an unreplicated client and the gateway, (b) from the gateway to a replicated object within the fault tolerance domain, and (c) between replicated objects within the fault tolerance domain

that the server-side ORB publishes automatically contains the {gateway_host, gateway_port}. This eliminates the effort of having to parse the IOR string to do the replacement, and also results in fewer undesirable interceptions. The gateway_host and the gateway_port are dedicated choices that are supplied to the interceptor at system configuration time.

When an unreplicated client uses this IOR, the client-side ORB, implicitly assuming that the host and port in the IOR refer to the server object, connects the client to the gateway. The gateway now becomes the recipient of every IIOP message sent by the unreplicated client, which continues to "believe" that the gateway is indeed the target server object. By extracting the server's object key (which the client-side ORB inserts into IIOP invocations to identify the target server), the gateway identifies the target server, multicasts the client invocation to the server object group. The gateway inserts sufficient information into the multicast messages to enable it to associate the server's response with the client's invocation.

The gateway process must be continuously listening for connections from unreplicated clients on its dedicated {gateway_host, gateway_port}. For each new client that contacts the gateway, the gateway spawns a new TCP/IP socket to communicate solely with that client, and uses the original socket to listen for further clients. The additional spawned sockets are destroyed when the connection between the unreplicated client and the gateway terminates.

Note that the replacement of the {server_host, server_port} in the IOR does not affect connection establishment or communication within the fault tolerance domain. Replicated clients wishing to communicate with replicated servers within the fault tolerance domain never use this TCP/IP-specific addressing in-

formation, but use instead the server's object group identifier to contact the replicated server through the fault tolerance infrastructure.

3.2 Encapsulation of IIOP into Multicast Messages

A gateway must encapsulate the IIOP invocations from the external unreplicated clients into multicast messages for transmission to the target replicated server object within the fault tolerance domain. Similarly, the IIOP responses, encapsulated within the multicast messages returned by the replicated server object, must be extracted by the gateway and returned to the unreplicated clients.

When an IIOP-encapsulating message is multicast by the gateway into the fault tolerance domain, the message contains the gateway_group_id as the sender group, and the server_group_id (determined by the gateway from the server's object key embedded in the client's IIOP invocation) as the destination group. The message is received in total order by the Replication Mechanisms hosting each of the server replicas. The replicated server performs the operation, and the fault tolerance infrastructure multicasts the results to the gateway. The replicated server assumes that the gateway that sent the IIOP invocation is a CORBA client object. Eternal's transparency through interception effectively ensures that neither the unreplicated client, nor any of the server replicas, is ever aware of communicating through the fault tolerance infrastructure using reliable multicast. The gateway (and, of course, the fault tolerance infrastructure itself) is the only party in the chain of communication that is aware of the reliable multicast and the fault tolerance infrastructure.

When the replicated server returns the response to the gateway, the IIOP response from each server replica is encapsulated by the Replication Mechanisms hosting that replica into a multicast message. The message contains the server_group_id as the sender group, and the gateway_group_id as the destination group. This information is insufficient for the gateway to route the IIOP response to the client replica that invoked the operation because multiple unreplicated TCP/IP-based clients may have invoked the same replicated server through the gateway. The gateway has no way of discriminating between these clients.

Thus, every multicast message must contain additional information, inserted by the gateway to identify each TCP/IP client that contacts the gateway. The resulting multicast messages have the structure shown in Figure 4. For every multicast message exchanged between replicated objects within the fault tolerance domain, the TCP/IP client identification is set to some unused value. The gateway (as well as the fault tolerance infrastructure) uses the destination group identifier, the source group identifier and the TCP/IP client identifier *collectively* to route every message to its intended destination.

Ideally, the client identification information ought to be supplied by the client-side ORB, as discussed in Section 3.5. Because this is not the case with current ORBs, the gateway maintains a simple counter, one for each destination server group. For each incoming TCP/IP client, the gateway first determines the server_group_id from the first IIOP message received from the client. The

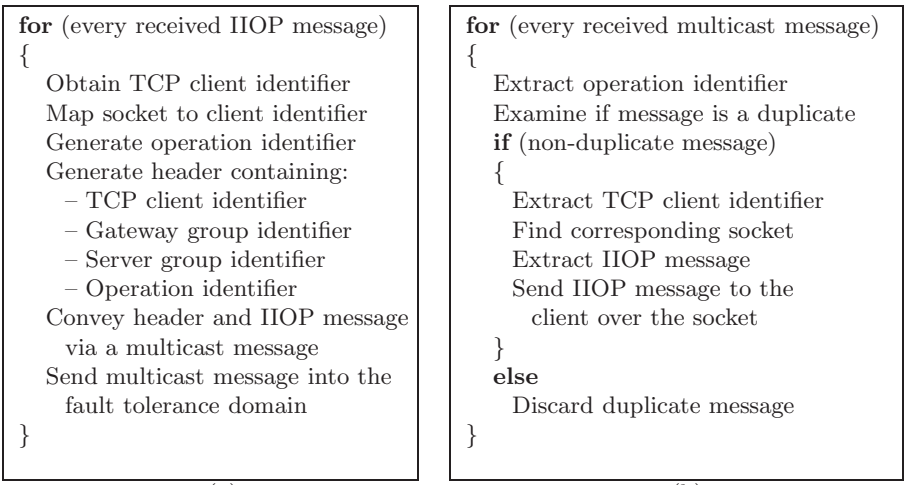

Fig. 5. Actions of the gateway for incoming messages from (a) external unreplicated clients outside the fault tolerance domain, and (b) replicated objects within the fault tolerance domain

gateway then uses the value of the counter corresponding to that server group as the TCP/IP client identifier. The counter is then incremented, to serve as the identifier for the next TCP/IP client for the same replicated server. The disadvantage of the gateway-assigned client identifiers, over identifiers supplied by the client-side ORB, is discussed in Section 3.4.

Figure 5 shows the sequence of steps that the gateway executes for incoming IIOP messages from outside the fault tolerance domain, and incoming multicast messages from within the fault tolerance domain.

3.3 Duplicate Detection and Suppression

To ensure replica consistency, duplicate detection and suppression mechanisms are used by Eternal throughout the fault tolerance domain; the gateways also employ these mechanisms for filtering duplicate responses from the replicated server objects within the fault tolerance domain. The gateway returns only a distinct copy of each response to the invoking external client. The duplicate copies of each response, if not suppressed, would be delivered to the client object, and may cause the client object's state to be corrupted.

To detect duplicate copies of each response, both the fault tolerance infrastructure and the gateway prepend an *operation identifier* to each message that is multicast within the fault tolerance domain, as shown in Figure 4. The operation identifier takes the form of either an *invocation identifier* for all multicast messages that encapsulate IIOP invocations, or a *response identifier* for all messages that encapsulate IIOP responses.

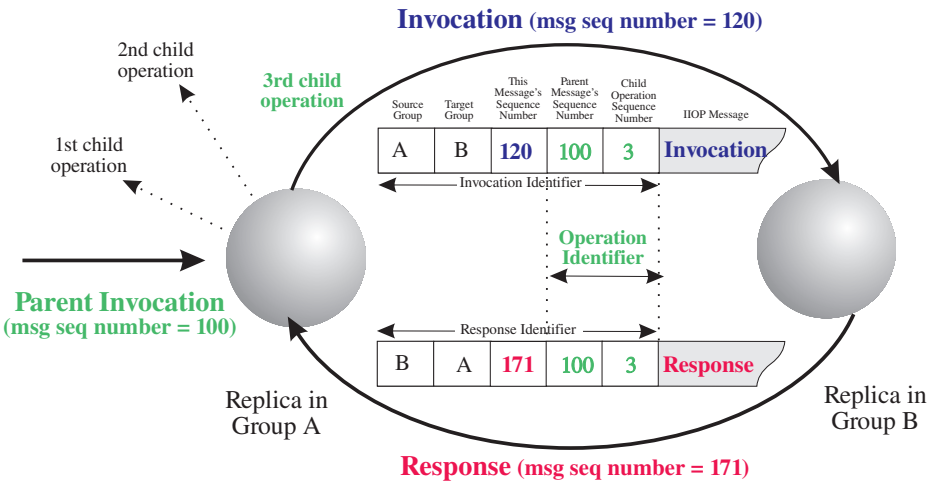

Fig. 6. Assignment of invocation, response and operation identifiers

Operation Identifiers For each outgoing IIOP invocation received by the gateway from an unreplicated client, the gateway generates the invocation identifier as shown in Figure 6. The gateway then inserts the invocation identifier into the Eternal-specific header of the message that it multicasts into the fault tolerance domain. The timestamp of this multicast message, which forms a part of the invocation identifier, is filled in by the fault tolerance infrastructure at the receiving end, when the message is delivered. The timestamp information is derived from the totally-ordered message sequence numbers assigned by the multicast group communication system.

For each outgoing IIOP response sent by a server replica, Eternal "remembers" and reuses a portion of the invocation identifier that was sent with the corresponding invocation. The portion of the invocation identifier that is reused in its counterpart response identifier is the *operation identifier*, which completely and uniquely identifies the operation consisting of the invocation-response pair. Both the invocation and the response identifiers have the same operation identifier fields. Furthermore, the operation identifier is identically determined at every server replica.

The gateway, on receipt of multiple copies (one copy for each server replica that returns a response) of a response to an IIOP invocation that it multicasts into the domain, can deliver the first copy that it receives, and discard all subsequently received copies by simply comparing the response identifier fields of the Eternal-specific header.

An invocation identifier has the form $(T_{B_{inv}}, (T_{A_{inv}}, S_{A_{inv}}))$, where $T_{A_{inv}}$ is the timestamp of the message containing the invocation of group A, $T_{B_{inv}}$ is the timestamp of the message containing the invocation of the group B, and $S_{A_{inv}}$ is the sequence number of the invocation of B in the sequence of invocations

by group A. Similarly, a response identifier has the form $(T_{B_{res}}, (T_{A_{inv}}, S_{A_{inv}}))$, where $T_{B_{res}}$ is the timestamp of the message containing the response by group B to group A and the other two fields are the same as for the invocation identifier. These invocation and response identifiers are contained in the multicast messages.

Note that the timestamps $T_{A_{inv}}$, $T_{B_{inv}}$ and $T_{B_{res}}$ are derived from the totally-ordered message sequence numbers assigned by the Totem multicast group communication system. The system-wide uniqueness of these timestamps (as a result of the total ordering) contributes to the uniqueness of the operation identifiers, and thus, to the detection of duplicate messages.

In the example of Figure 6, $T_{A_{inv}}$ corresponds to 100, $S_{A_{inv}}$ corresponds to 3, $T_{B_{inv}}$ corresponds to 120 and $T_{B_{res}}$ corresponds to 171. In the case of the gateway, A represents the gateway_group and B represents the target replicated server that the unreplicated client (that connects to the gateway) wishes to contact.

3.4 Using Existing ORBs

Existing ORBs do not have the capability to traverse a list of profiles, and select the next profile if the first one fails on connection. The disadvantage of this is that redundant gateways are not possible. Clients may experience disconnection if the processor hosting the gateway fails, and does not recover. The processor hosting the gateway is a single point of failure. If the client ORB has the capability to understand only the first IIOP profile (the standard TAG_INTERNET_IOP profile), and if the gateway to which it connects using the first profile fails, the client has no alternative but to abandon the request. Furthermore, the client does not know the status of any invocations that it has already sent, for which it is still awaiting responses.

An alternative to using multiple gateways might be to have a cold passively replicated gateway. In this case, the gateway's state should be checkpointed often enough to allow it to be recovered. However, clients will still be disconnected from the gateway if it fails, and must have mechanisms to allow them to reconnect to the gateway, when it recovers.

In the case of redundant gateways, the new gateway to which the client connects (on failure of the first gateway) has no way of "knowing" that this is the same client. The simple counter mechanism, described in Section 3.2, is insufficient in this case to identify the client. This means that, even if the new gateway receives the response for an outstanding invocation sent by the client through the first gateway, the new gateway does not know which of its connected clients should receive this response. Secondly, if the client were now to re-issue all of the pending invocations to the new gateway, the new gateway may, in turn, re-issue these invocations to the replicated objects within the fault tolerance domain, thereby corrupting their state.

Thus, due to lack of client-side identification provided by the ORB, the gateway cannot prevent duplication of client requests if

- The unreplicated client fails, recovers and resends its request (this is outside the fault tolerance domain's and the gateway's control, and cannot be handled without extending some of the fault tolerance mechanisms to the unreplicated client)
- The gateway process fails, and then recovers, and the client reconnects to the gateway
- Redundant gateways are used, and the original gateway fails, and the client switches to the next operational gateway

3.5 Enhancements to Existing ORBs

If only a single gateway is provided for a fault tolerance domain, it is insufficient to guarantee the level of reliability that customers of Internet-based applications have come to expect. For instance, if a customer uses an unreplicated Web browser to connect to a replicated stock trading server through a gateway, the failure of the gateway could leave the customer wondering about the status of any outstanding invocations issued on the stock trading server. Because the gateway constitutes a single point of failure, the benefits of the server replication are lost to the customer.

The use of redundant gateways requires additional intelligence on the part of the client-side ORB to exploit the multiple gateways. Unfortunately, the required mechanisms are not part of the current CORBA standard. In the absence of the required support in current ORBs, we have implemented a thin client-side interception layer that mimics the support that an enhanced client-side ORB would provide to allow unreplicated CORBA clients to benefit from fault tolerance. As discussed in Section 3.5, we envisage that the functionality of this interception layer will eventually incorporated into the client-side ORB itself.

According to the current CORBA standard, a profile contains addressing information within an IOR. An object's IOR can contain multiple profiles, with each profile designating an alternative address for contacting the object. To allow the addressing information for the multiple gateways to be made available to unreplicated clients, the Eternal Interceptor "stitches"' together the addressing information for each gateway into a single multi-profile IOR.

On the client side, the thin interception layer has the capability of traversing the profiles within the multi-profile IOR, should this be required. The interception layer connects the client object to the first gateway listed in the multi-profile IOR, and inserts a unique TCP/IP client identifier into the service context field (a part of the IIOP request and reply messages), where the user may insert information; if a receiving ORB cannot interpret this information, it will ignore it) of each IIOP message sent out by the client. The advantage of using the service context field is that it can be safely ignored (as is the case here) by a server ORB that does not understand it. It is intended purely for the consumption of the gateway.

For each IIOP request message that a gateway receives from a client, the gateway first multicasts the message to the group of gateways. This is done so that every gateway in the group has a record of the invocation in case the first

connected gateway fails. The gateway group then multicasts the message into the fault tolerance domain, and the gateway group (and not the connected gateway alone) receives the response.

If the first gateway fails to respond, the client-side interception layer transparently skips to the next profile in the multi-profile IOR, and connects the client to the next operational gateway, and reissues any pending invocations. If the client object sent an invocation for which a response was expected from the first gateway, the client-side interception layer obtains it from the next operational gateway. This is possible because the client-side interception layer supplies the same unique client identifier for each of its requests, along with a unique request identifier, which would make it possible for the new gateway to detect reinvocations due to reconnection of the client-side interception layer to a different gateway. The reason for the reinvocations is two-fold: firstly, it allows the client-side interception layer to communicate the client's unique identifier to the gateway, and secondly, the client-side interception layer has no way of knowing if the first invocation ever reached the original failed gateway. Each gateway also contains the intelligence to inform all of the other gateways in the event that the client fails. In this case, the gateways can delete any state that they may have stored on behalf of the client.

Eternal's duplicate detection and suppression mechanisms described in Section 3.3, along with the unique client identifier, and CORBA's existing request identifier mechanisms, enable the gateway to preserve the replica consistency within the fault tolerance domain, as well as to protect the unreplicated client outside the fault tolerance domain from having its state corrupted. Furthermore, the redundant gateways scheme enables the unreplicated client to benefit from the fault tolerance of the server.

4 Related Work

Other systems have addressed issues related to consistent object replication and fault tolerance for CORBA applications. The Object Group Service [3] provides replication for CORBA applications through a set of CORBA services. Replica consistency is ensured through group communication based on a consensus algorithm implemented through CORBA service objects. Mechanisms have been provided for duplicate detection and suppression, and for state transfer of application state.

The Maestro toolkit [12] includes an IIOP-conformant ORB with an open architecture that supports multiple execution styles and request processing policies. The replicated updates execution style can be used to add reliability and high availability properties to client/server CORBA applications in settings where it is not feasible to make modifications at the client side, as in the case of unreplicated clients contacting replicated server objects. The Maestro toolkit addresses some of the issues in the implementation of gateways.

The AQuA architecture [1,10] is a CORBA-based dependability framework that provides object replication and fault tolerance. AQuA exploits the group

communication facilities and the ordering guarantees of the underlying Ensemble and Maestro toolkits to ensure replica consistency for the application. The AQuA gateway translates CORBA object invocations into messages that are transmitted via Ensemble. Duplicate invocations and duplicate responses are detected and filtered by the gateway.

The Distributed Object-Oriented Reliable Service (DOORS) [11] provides fault tolerance through a CORBA-compliant service approach. DOORS consists of CORBA objects that detect, and recover from, replica and processor faults. The system provides support for management of resource and reliability requirements based on the needs of the CORBA application. DOORS employs libraries for the transparent checkpointing of applications; however, duplicate detection and suppression are not addressed.

5 Conclusion

The Eternal system allows applications to span multiple enterprises over the Internet, with the application being decomposed into fault tolerance domains, with the mechanisms of Eternal providing strong replica consistency within each fault tolerance domain. In addition, Eternal provides gateways to allow unreplicated clients and other fault tolerance domains to communicate with the replicated server objects within a fault tolerance domain, without compromising the replica consistency within any fault tolerance domain. Through the use of interception, Eternal provides this fault tolerance transparently to the CORBA application and to the ORB. The gateway mechanisms are crucial to today's applications, where clients are unreplicated, but nevertheless wish to benefit from the fault tolerance provided for the servers.

References

1. M. Cukier, J. Ren, C. Sabnis, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr, and R. Schantz. AQuA: An adaptive architecture that provides dependable distributed objects. In *Proceedings of the IEEE 17th Symposium on Reliable Distributed Systems*, pages 245–253, West Lafayette, IN, October 1998. 101
2. Eternal Systems and Sun Microsystems. Fault tolerant CORBA using entity redundancy: Initial joint submission. OMG Technical Committee Document orbos/98-04-08, October 1998. 91
3. P. Felber, R. Guerraoui, and A. Schiper. The implementation of a CORBA object group service. *Theory and Practice of Object Systems*, 4(2):93–105, 1998. 101
4. 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 the ACM*, 39(4):54–63, April 1996. 90
5. L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan. Consistent object replication in the Eternal system. *Theory and Practice of Object Systems*, 4(2):81–92, 1998. 90
6. P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith. Replica consistency of CORBA objects in partitionable distributed systems. *Distributed Systems Engineering*, 4(3):139–150, 1997. 90

7. P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith. Using interceptors to enhance CORBA. *IEEE Computer*, pages 62–68, July 1999. 91
8. Object Management Group. Fault tolerant CORBA using entity redundancy: Request for proposals. OMG Technical Committee Document orbos/98-04-01, April 1998. 89, 91
9. Object Management Group. The Common Object Request Broker: Architecture and specification, 2.3 edition. OMG Technical Committee Document formal/98-12-01, June 1999. 89
10. R. Schantz, J. Zinky, D. A. Karr, D. Bakken, J. Megquier, and J. Loyall. An object-level gateway supporting integrated-property quality of service. In *Proceedings of the IEEE 2nd International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 223–234, Saint Malo, France, May 1999. 101
11. J. Schonwalder, S. Garg, Y. Huang, A. P. A. van Moorsel, and S. Yajnik. A management interface for distributed fault tolerance CORBA services. In *Proceedings of the IEEE 3rd International Workshop on Systems Management*, pages 98–107, Newport, RI, Apr. 1998. 102
12. A. Vaysburd and K. Birman. The Maestro approach to building reliable interoperable distributed applications with multiple execution styles. *Theory and Practice of Object Systems*, 4(2):73–80, 1998. 101