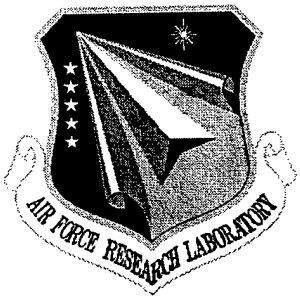
AFRL-IF-RS-TR-2001-116
Final Technical Report
June 2001



THE ETERNAL SYSTEM

University of California at Santa Barbara

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. F161

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2001-116 has been reviewed and is approved for publication.

APPROVED: Patrick M. Hurley

PATRICK M. HURLEY
Project Engineer

FOR THE DIRECTOR: Warren

WARREN H. DEBANY, Technical Advisor
Information Grid Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFGA, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

THE ETERNAL SYSTEM

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

Contractor: University of California at Santa Barbara
Contract Number: F30602-97-1-0284
Effective Date of Contract: 10 January 1997
Contract Expiration Date: 30 September 2000
Short Title of Work: The Eternal System
Period of Work Covered: Jan 97 - Sep 00

Principal Investigator: L. E. Moser
Phone: (805) 893-4897
AFRL Project Engineer: Patrick M. Hurley
Phone: (315) 330-3624

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Patrick M. Hurley, AFRL/IFGA, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JUNE 2001	Final Jan 97 - Sep 00

4. TITLE AND SUBTITLE
THE ETERNAL SYSTEM

5. FUNDING NUMBERS
C - F30602-97-1-0284
PE - 62301E
PR - F161
TA - 40
WU - 32

6. AUTHOR(S)
L. E. Moser and P. M. Melliar-Smith

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of California at Santa Barbara
Department of Electrical and Computer Engineering
Santa Barbara CA 93106

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

Air Force Research Laboratory/IFGA
26 Electronic Parkway
Rome New York 13441-4514

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2001-116

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Patrick M. Hurley/IFGA/(315) 330-3624

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

UCSB received a 3 year grant in October 1997, to explore how distributed object applications can perform live upgrades and be made fault-tolerant by replicating their constituent objects, and distributing these replicas across different computers in the network. The technology of Eternal was submitted in response to the October 1998 Object Management Group's Request for Proposals on Fault-Tolerant CORBA. A significant body of work exists in the area of fault-tolerant distributed object systems; much of this work uses object replication to provide fault tolerance. This project was different in that it focused on the degree of transparency to the CORBA application, the degree of modification to the CORBA ORB, the specific mechanisms for achieving replica consistency, and the level of replica consistency provided. Previous efforts to enhance CORBA with fault tolerance attempted to embed fault tolerance mechanisms within the ORB itself. The novel interception approach, developed with this work, allows the transparent insertion of fault tolerance mechanisms underneath the ORB. The interception approach involves "capturing" specific system calls or library routines used by the application, and modifying their call parameter or return values, or even the calls and routines themselves, to alter the behavior of the application. the advantages to this approach are that neither the ORB nor the objects are ever aware of being "intercepted" and thus, the new functionality is provided to the application in a manner that is transparent both to the application and to the ORB.

14. SUBJECT TERMS
Adaptable, Survivable, Distributed System, Object-Oriented, OMG, CORBA, Live Upgrades, Replication

15. NUMBER OF PAGES
48

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

LIST OF FIGURES

The Eternal System
Final Report, May 2000

L. E. Moser and P. M. Melliar-Smith
Department of Electrical and Computer Engineering
University of California, Santa Barbara 93106
(805) 893-4897 and (805) 8934-8438
moser@ece.ucsb.edu and pmms@ece.ucsb.edu

1 Significant Accomplishments

The objective of this project was to develop capabilities for fault tolerance and live software upgrades for application systems developed within the CORBA distributed programming model. The project has been a remarkable success, and has resulted in:

- Design and implementation of the CORBA-based Eternal system
- Adoption of a new industry standard for Fault Tolerant CORBA
- Formation of a startup company to commercialize the technology developed in this project.

The project has also resulted in the design of the Java-based Aroma system, which has similar functionality to that of Eternal. It is very satisfying that this DARPA-funded research has proceeded so rapidly to a successful demonstration, standardization and commercial products.

1.1 Challenges for Consistent Replication

The Eternal system provides fault tolerance for CORBA applications by replicating the application objects. The purpose of replication is to provide multiple, redundant, identical copies, or *replicas*, of an object so that the object can continue to provide useful services, even if some of its replicas fail, or the processors hosting some of its replicas fail. For replication to work correctly, all of the replicas of an object must have consistent state, under both fault-free and recovery conditions.

To ensure *strong replica consistency* of the application, Eternal requires application objects to be *deterministic* in their behavior so that if two replicas of an object start from the same initial state, and have the same sequence of messages applied to them, in the same order, the two replicas will reach the same final state. Eternal provides mechanisms that address the challenges in maintaining replica consistency:

- **Ordering of Operations.** All of the replicas of each replicated object must perform the same sequence of operations in the same order to achieve replica consistency. Eternal achieves this by exploiting a reliable totally ordered multicast group communication system for conveying the IIOP invocations (responses) to the replicas of a CORBA server (client).
- **Duplicate Operations.** Replication, by its very nature, may lead to duplicate operations. For example, when every replica of a three-way actively replicated client object invokes a method of a replicated server object, every server replica will receive three copies of the same invocation, one from each of the client replicas. Eternal ensures that such duplicate invocations (responses) due to a replicated client (server) object are filtered so that the server (client) object receives only a single copy of every distinct invocation (response).
- **Recovery.** When a new replica is activated, or when a failed replica is recovered, *before* it issues an invocation, performs an operation, or issues a response, it must have the same state as the other replicas of the object. Eternal provides mechanisms for retrieving the state from an existing operational replica of the object and transferring the state to the new or recovering replica.
- **Multithreading.** Many commercial ORBs and CORBA applications employ multithreading, a significant source of non-deterministic behavior. Replicas of a multithreaded object might become inconsistent if the threads, and the operations that they execute, are not carefully controlled. For multithreaded ORBs that allow an object to execute multiple operations simultaneously, Eternal provides mechanisms to ensure replica consistency, regardless of the multithreading of the ORB or the application.

Eternal's OMG-compliant fault tolerance infrastructure consists of CORBA objects *above* the ORB and mechanisms *underneath* the ORB, as shown in Figure 1. Because the components above the ORB are CORBA objects, they can also be replicated, with a sufficient number of replicas distributed across

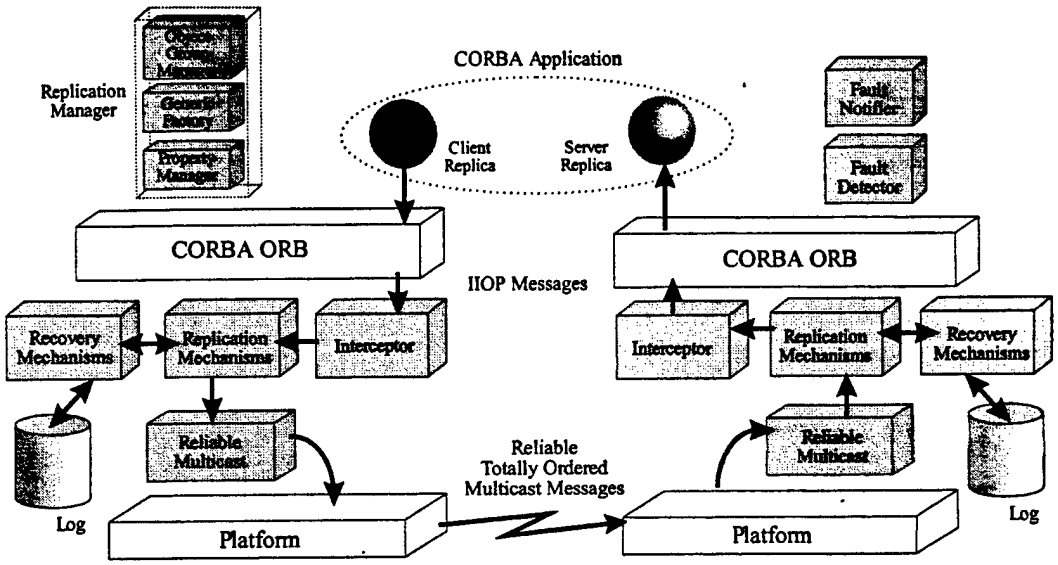


Figure 1: Eternal's OMG-compliant fault tolerance infrastructure.

the system. Eternal's mechanisms, on the other hand, must be present on every processor within the fault tolerance domain.

1.2 Replication Management

To manage the replication of an object, Eternal employs the notion of an object group, where the members of the group correspond to the replicas of an object. In Eternal, both client and server objects can be replicated and, thus, constitute object groups.

The Replication Manager is a crucial component of the infrastructure, and handles the creation, deletion and replication of both the application objects and the infrastructure objects within the fault tolerance domain. The Replication Manager replicates objects, and distributes the replicas across the system. Although each replica of an object has an individual object reference, the Replication Manager fabricates an object group reference for the replicated object that clients use to contact the replicated object. Through inheritance, the Replication Manager incorporates the functionalities of the Property Manager, Generic Factory and Object Group Manager.

The Property Manager allows a user to assign values to a number of fault

tolerance properties for every application object that is to be replicated. Eternal provides the user with the flexibility to configure the replication of every application object by assigning the values of the following fault tolerance properties:

- **Replication Style** – stateless, actively replicated, cold passively replicated or warm passively replicated.
- **Membership Style** – addition of replicas to, or removal of replicas from, the object group is application-controlled or infrastructure-controlled.
- **Consistency Style** – replica consistency (including replication, recovery, checkpointing, logging, etc.) is application-controlled or infrastructure-controlled.
- **Factories** – objects that create and delete the replicas of the object.
- **Initial Number of Replicas** – the number of replicas of the object to be created initially.
- **Minimum Number of Replicas** – the number of replicas of the object that must exist for the object to be sufficiently protected against faults.
- **Checkpoint Interval** – the frequency at which the state of an object is to be retrieved and logged for the purposes of recovery.

The Generic Factory allows users to create replicated objects in the same way that they would normally create unreplicated objects. This interface is inherited by the Replication Manager to allow the application to invoke the Replication Manager directly to create and delete replicated objects. When asked to create a replicated object through its Generic Factory interface, the Replication Manager, in turn, delegates the operation to the factories on the processors where the individual replicas of the object are to be created.

The Object Group Manager allows users to control directly the creation, deletion and location of individual replicas of an application object. While this violates replication transparency (because the user is explicitly aware of the replicas of an object), and must be used with care so that replica consistency is maintained, it is useful for expert users who wish to exercise direct control over the replication of application objects.

Infrastructure-controlled Membership Style in conjunction with infrastructure-controlled Consistency Style is favored for the development of fault-tolerant CORBA applications, because it provides the maximal ease of use and transparency to the application, with the assurance of strong replica consistency, under both fault-free and recovery conditions.

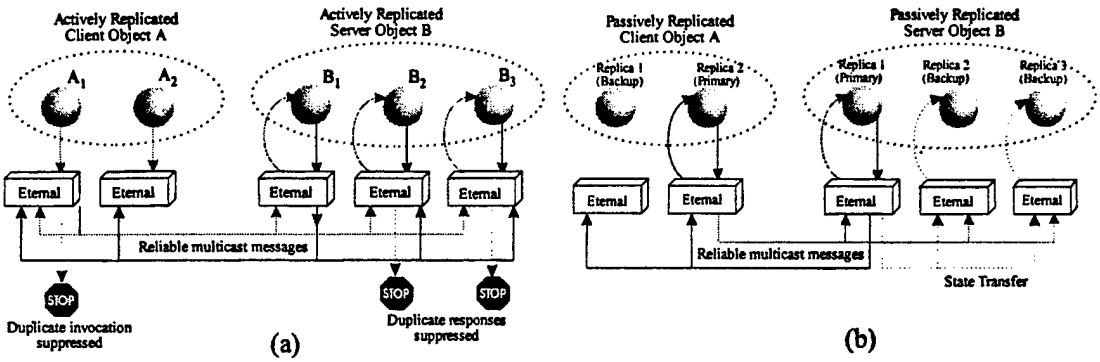


Figure 2: Two of the Replication Styles supported by the Eternal system (a) active replication and (b) warm passive replication.

1.3 Transparency via Interception

Eternal's Interceptor is a non-ORB-level, non-application-level component that transparently "attaches" itself to every CORBA object at runtime, without the object's or the ORB's knowledge, and that can modify the object's behavior as desired. The Interceptor ensures that the application's IIOP messages (containing the client's invocations and the server's responses), originally destined for TCP/IP, are diverted instead to the Replication Mechanisms. The advantage of the Interceptor, located underneath the ORB, is not only its transparency to the ORB and to the application, but also its implementation in an ORB-independent manner.

Eternal's Interceptor currently employs the library interpositioning hooks found on Unix and Windows NT. Library interpositioning involves the transparent runtime replacement of the socket-level library routines used by the CORBA application for connection establishment over TCP/IP. Eternal's Interceptor captures, and redefines, those routines so that the application's TCP/IP connections are transparently converted into connections to the Replication Mechanisms. Once the connections are established, the Interceptor adds no overhead in the path of message communication because the application automatically (and unknowingly) uses the connection to the Replication Mechanisms to send IIOP messages.

1.3.1 Replication Mechanisms

To facilitate replica consistency, the Replication Mechanisms convey the IIOP messages of the CORBA application using the reliable totally-ordered multi-

cast messages of the underlying Totem multicast group communication protocol.

Eternal's Replication Mechanisms perform different operations for the different replication styles, as shown in Figure 2. For an actively replicated server (client) object, each replica responds to (invokes) every operation. Thus, the Replication Mechanisms deliver every IIOP invocation (response) intended for a replicated server (client) to every server (client) replica through the Interceptor. For active replication, the failure of a single active replica is masked due to the presence of the other active replicas that are also performing the operation.

For a passively replicated server (client) object, only one of the replicas, designated the primary, responds to (invokes) every operation. In this case, the Replication Mechanisms deliver every IIOP invocation (response) only to the primary server (client) replica. In the case of warm passive replication, the backup replicas are synchronized periodically with the primary replica. In the case of cold passive replication, the backup replicas are not loaded, but Eternal periodically retrieves, and stores in a log, the state of the primary replica. In the event that the primary replica fails, one of the backup replicas takes over as the new primary replica.

1.4 Fault Detection and Notification

The Fault Detector is capable of detecting host, process and object faults. Each application object must inherit a monitorable interface to allow the Fault Detector to determine the object's status. The Fault Detector communicates the occurrence of faults to the Fault Notifier.

On receiving reports of faults from the Fault Detector, the Fault Notifier filters them to eliminate any inappropriate or duplicate reports. The Fault Notifier then distributes fault event notifications to all of the objects that have subscribed to receive such notifications. The Replication Manager is one such subscriber.

Eternal allows the user to influence fault detection for an object through the following fault tolerance properties:

- **Fault Monitoring Style** – the object is monitored by periodic "pinging" (pull monitoring) of the object, or by periodic "i-am-alive" messages (push monitoring) sent by the object.
- **Fault Monitoring Granularity** – the replicated object is monitored on the basis of a replica, a location, or a location-and-type.

- **Fault Monitoring Interval** – the frequency at which an object is to be "pinged" to detect if it is alive or has failed.

1.5 Logging and Recovery

Every replicated CORBA object can be regarded as having three kinds of state: *application state* (known to, and programmed into the object by, the application programmer), *ORB state* (maintained by the ORB for the object) and *infrastructure state* (invisible to the application programmer and maintained for the object by Eternal). Application state is typically represented by the values of the data structures of the replicated object. ORB state is vendor-dependent and consists of the values of the data structures (last-seen request identifier, threading policy, *etc.*). Infrastructure state is independent of, and invisible to, the replicated object as well as the ORB, and involves information that Eternal maintains for consistent replication.

Eternal's Recovery Mechanisms ensure that all of the replicas of an object are consistent in application, ORB and infrastructure state. The Recovery Mechanisms handle the restoration of a new primary replica's state, as well as the periodic retrieval of an operational primary replica's state. The transfer of state to a new or recovering replica includes the transfer of application state to the new replica, ORB state to the ORB hosting the new replica, and infrastructure state to the Recovery Mechanisms that manage the new or recovered replica. To enable application state to be captured and logged for the purposes of recovery, every replicated CORBA object must inherit the `Checkpointable` interface that contains methods for retrieving (`get_state()`) and assigning (`get_state()`) an object's state.

Because state retrieval from an existing active (primary passive) replica occurs at a different point in the message sequence from the assignment of the retrieved state to the new active (backup passive) replicas, the Recovery Mechanisms at the state retrieval and assignment locations synchronize the retrieval and state assignment messages. Furthermore, the Recovery Mechanisms log all new invocations and responses that arrive for a replica while its state is being assigned for delivery after the state assignment is complete.

To enable incoming response messages to be matched with their corresponding invocations, and to ensure that the target application objects receive only one copy of every distinct invocation or response intended for them, the Recovery Mechanisms insert an operation identifier into the Eternal-specific header for each outgoing IIOP message.

By deriving operation identifiers from the unique totally-ordered sequence

numbers assigned by the underlying multicast group communication protocol to each message that it delivers, the Recovery Mechanisms on *different* processors assign the same operation identifier for a specific operation. Thus, if a three-way replicated client invokes an operation, the three duplicates (one from each replica) of the same invocation will carry the same operation identifier in the Eternal-specific header; distinct invocations are assigned distinct operation identifiers. At the Recovery Mechanisms hosting the target server replicas, the first of the three invocations to arrive is delivered to the server; the examination of the operation identifiers of the subsequently received duplicates leads to their suppression.

1.6 Implementation and Performance

The current implementation of Eternal is capable of providing transparent fault tolerance to unmodified applications running over unmodified commercial ORBs (VisiBroker, Orbix, TAO, ORBacus, e*ORB, CORBAplus, omniORB2 and ILU), over standard operating systems (Solaris 2.x, Red Hat Linux 6.0 and HP-UX 10.20).

To measure the performance of Eternal for the different replication styles and levels of fault tolerance, we used a simple test application developed with the VisiBroker 3.2 ORB. The measurements were taken over a network of six dual-processor 167 MHz UltraSPARC workstations, running the Solaris 2.5.1 operating system and connected by a 100 Mbps Ethernet. The graph in Figure 3 shows the throughputs obtained with this test application for the following cases, which are listed in the order of increasing level of fault tolerance:

- **Case 1:** Unreplicated client and server objects without the Eternal system. The throughput is determined only by the ORB mechanisms.
- **Case 2:** Three-way active replication of both client and server objects without majority voting. Reliable totally ordered multicasts without either the message digests or the signatures are used. The throughput is dictated by the cost of interception, active replication and multicasting, in addition to the costs for case 1. The Totem protocol is employed in this case.
- **Case 3:** Three-way active replication of both client and server objects with majority voting. Secure reliable totally ordered multicasts with message digests are used. The throughput is dictated by the cost of message digests, in addition to the costs for case 2. The SecureRing

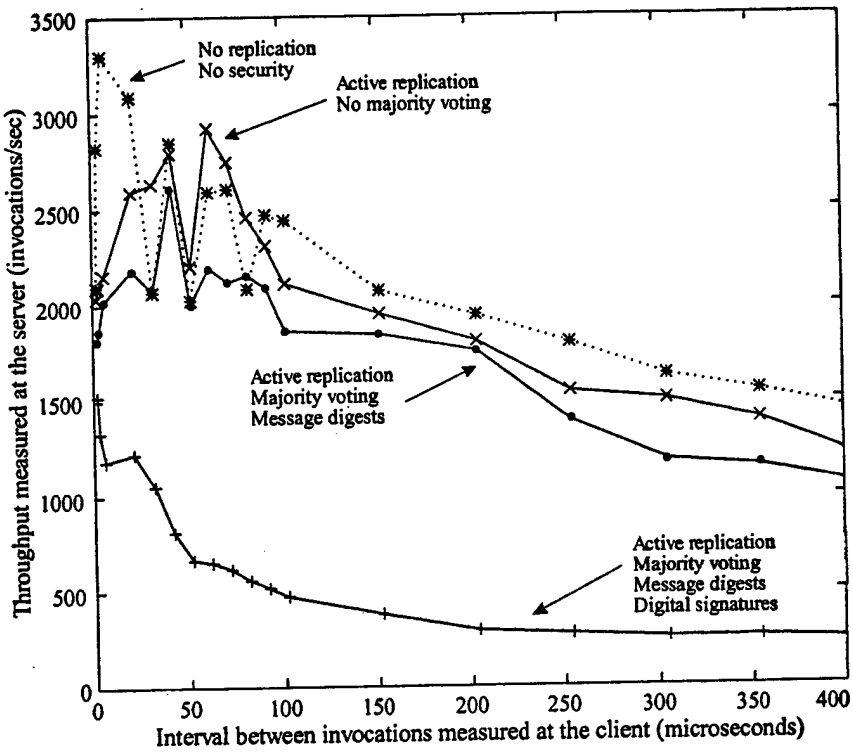


Figure 3: Performance of the Eternal system.

protocol is employed in this case.

- **Case 4:** Three-way active replication of both client and server objects with majority voting. Secure reliable totally ordered multicasts with message digests and digitally signed tokens are used. The throughput is dictated by the cost of signatures, in addition to the costs for case 3. The SecureRing system is employed in this case.

For cases 2, 3 and 4, up to six multicast messages were sent with each token visit, where each multicast message encapsulates possibly multiple IIOP messages. While the cost of computing a single signature is spread over six messages, the use of digital signatures is nevertheless computationally expensive, as can be seen from the overheads of the Eternal system in case 4. However, the results indicate that the overheads of the Eternal system without signatures (cases 2 and 3) are low. In particular, the overheads are in the range of 7-15% for remote invocations for the triplicated clients and the triplicated servers of case 2. In all of the cases, the overheads are quite reasonable.

1.7 Live Software Upgrades

In the Eternal system, objects are replicated to provide fault-tolerance. The Eternal Evolution Manager exploits this replication to support upgrades of CORBA application objects. It is assumed that the application programmer writes the new version of the application code himself. The Eternal Evolution Manager performs offline analysis necessary for live upgrades, and it automatically performs the sequence of operations necessary to perform the actualy upgrade.

We achieve upgrades without interrupting the executing application by performing a series of individual replacements which nudge the application towards an upgraded state but which do not affect the behavior of the program. We use automatically-generated intermediate objects that contain new objects with new interfaces that coexist with old objects still using their old interfaces. The replicas being upgraded are replaced, one at a time, by their intermediate versions, which continue to execute the old methods. Once all of these intermediate versions are in place, we effect an atomic switchover after which only the new methods are invoked. Then, to clean up the program, we replace the intermediate objects one at a time with final versions containing only the new version of the code, all the while executing the new methods.

When an object's interface changes as the result of an upgrade, the situation becomes more complex. Suppose that the upgrade of B involves a signature change in method B.m(). If we were to upgrade only B, and ignore the other application objects, any other object that attempts to invoke B.m() would cause an error. Therefore, we use coordinated upgrade sets – groups of objects that are evolved "together" – to upgrade multiple objects at the same time.

We have demonstrated interface preserving and interface changing upgrades running independently of the rest of the Eternal System. Specifically, we started several objects at the same time so that they appeared to be replicas of each other, and we upgraded these pseudo-replicas. We are continuing to work on integrating the Evolution Manager with the rest of the Eternal system so that real replicas can be upgraded.

1.7.1 Interface Comparisons

As discussed in previous reports, there are two basic types of CORBA object upgrades. The easier of the two occurs when the applications' IDL files

(which define the interfaces between CORBA objects) do not change. In this case, we do not need to check how the changes in one object's interface propagate to the other objects; the code modifications are confined to the particular object. The more difficult scenario occurs when an object's interface changes. This sort of upgrade may require modification of other objects that use this interface. The mechanics of the upgrade process depend largely on which type of upgrade is being performed. Therefore, the first type of code analysis that we completed determines whether changes have occurred between IDL files. We want to be able to compare two IDL files and determine whether the interfaces that they describe are equivalent. A simple diff old.idl new.idl is inadequate. For example, a diff would indicate that these two interfaces are different when, in fact, they are equivalent.

```
interface hat {
readonly attribute short size;
readonly attribute short color;
}

interface hat {
readonly attribute short size, color;
}
```

To recognize the equivalence of interfaces such as this, the IDL files are parsed with the JavaCC compiler (which conveniently contains a CORBA-IDL grammar in its free distribution package). The grammar contains statements such as:

```
void interface_header() :
{}
{
"interface" identifier() [ inheritance_spec() ]
}
}
```

We are able to add Java to this source code, so that certain operations are performed whenever an appropriate part of the grammar is encountered. For example, whenever we encounter an interface_header, we would like to create a new Java InterfaceObj in which we will store information about the interface's methods and attributes, as well as the name of the interface which this object represents. We accomplish this code by modifying the code above.

```
void interface_header() :
{
String interfaceName;
}
{
"interface" interfaceName = identifier() [ inheritance_spec() ]
{
theInterface = new interfaceObj(interfaceName);
}
}
```

There are similar methods which parse attribute and method declarations and code is similarly added to store methods and attributes in InterfaceObj. Comparing two IDL files now becomes quite easy. You just run each IDL file through the parser, and then compare the resulting data structures using the overloaded equality operator.

1.7.2 Automatically-Generated Intermediate Code

We have also implemented a tool that generates automatically the intermediate code necessary for live upgrades, because generating it by hand is quite tedious and error-prone.

For example, if the old and new versions of the CountObj code have the following method:

```
void UpdateCount(int updatedCountVal)
```

the following intermediate version would be generated automatically. It contains a member variable switchFlag, indicating if the switchover has occurred, as well as both the old and new versions of the method updateCount().

```
class CountObj_inter {
unsigned short switchFlag;
CountObj *oldCount;
CountObj_new *newCount;
...
void UpdateCount(int updatedCountVal) {
if (switchFlag == 0)
oldCount->UpdateCount(updatedCountVal);
else
```

```
newCount->UpdateCount(updatedCountVal);
}
}
```

We have written additional code that determines where the implementations of code versions differ.

1.7.3 Automatically-Generated State Transfer Code

Relying heavily on the JavaCC parser package, we now have the ability to automatically generate state transfer code to be used with generic C++ applications. The methods generated are char *SendState() and Receive State(char *theState). char *SendState returns a formatted character string, encapsulating the state of the object, and ReceiveState(char *) assigns the state from this character string.

All of the member variables are accounted for in this state transfer code. We assume that state will be transferred only between method invocations; consequently, we do not worry about local variables (those declared within method bodies). This assumption not only simplifies our code, but also is necessary to preserve replica consistency. Method invocation boundaries provide a convenient point of synchronization. We have no way of ensuring (with our current implementation) that replicas are executing identical parts of methods at the same time. Therefore, if we were to transfer state while a method was executing, different replicas would likely transfer different states, and replica consistency would be lost.

Inevitably, there will be cases in which the programmers wish to modify the way in which we transfer state. If an object contained a member variable char *hostname and state was being transferred to an object on a different host, some other mechanism (presumably a call to gethostname()) would be required to properly initialize the variable.

We have designed a GUI to achieve this level of user-interaction with the generation of the state transfer code. The GUI allows you to delete certain member variables, or if a member variable is a non-basic type, portions of a member variable from the state. The GUI additionally allows you to modify the automatically generated state transfer code.

Pointers

Handling pointers is somewhat more complicated. Consider the following example from the automatically-generated SendState code:

```
int testInt = 10;
int *pointerInt = &testInt;
```

Before properly handling pointers, the state we would have transferred would have looked like:

```
wrongstate[i] = "int;testInt;10"
wrongstate[i+1] = "int*;pointerInt;10"
```

With proper pointer handling, the state now looks like

```
correctstate[i] = "int;testInt;10"
correctstate[i+1] = "int*;pointerInt;LABEL=i"
```

More complicated examples include pointers that do not point to another variable (double *ptrDouble = new double(15);), forward pointers (pointers that point to variables not yet declared) and pointers to non-user-defined types.

Cycles

Once we started handling pointers, we had to deal with cyclic pointers. Consider the following example:

```
Class cycleTest {
cycleTest *back;
cycleTest *forward;
}

cycleTest a, b;

a.back = &b;
b.forward = &a;
```

A naive SendState method would encounter a, follow its back pointer to b, follow b's forward pointer to a, follow a's back pointer to b, and so on. A more realistic example of a cycle like this is a linked list, where the cycle is of arbitrary length. We would like to be able to detect a cycle the first time a variable occurs for the second time.

We handle this problem by keeping track of the pointer's address itself, in addition to the address to which the pointer points. The pointer's addresses are stored in a hash table, and as soon as a duplicate is detected, the state transfer code backs and moves on to the next variable.

1.7.4 State Conversion Code

Normally, state transfer is performed between objects of identical type, which therefore have the same state variables. However, the work on state transfer originally began as a spin-off from work on the Evolution Manger, so this assumption was never made. When an upgrade is taking place, and the state of an old version of an object is transferred to a new version of the object, the old object's state might not correspond to the state variables present in the new object.

Initially, we thought about creating a ConvertState(char *) method which would be invoked between char *SendState() and ReceiveState(char *) to deal with this problem. It was decided, however, that this additional level of indirection would slow down the state transfer mechanism. Instead, we modify a copy of the char *SendState() code that the earlier mechanisms generated. If member variables from the old code are no longer present in the new code, their presence in the char *SendState() code is simply deleted. If the new version includes a new member variable, initialization code (which is elicited from the user through another GUI) is incorporated into the char *SendState() code.

More difficult to handle is the case in which two member variables represent the same concept, but have different implementations. For example, in an old version of code, information might be stored in a linked list, while in a new version it is stored in a stack. The ability to automatically produce code that handles conversion between similar yet dissimilar member variables is currently beyond the scope of the project. So for now the user must provide the conversion code by interacting with a GUI.

1.8 Resource Management

Resource management in Eternal is implemented as a collection of CORBA objects which can be replicated and, thus, benefits from Eternal's fault tolerance. The Profilers receive information from the Replication Manager that includes the names of the methods invoked, the invoking methods, the name of the processors where the objects are located, and the time of the invocations. For each local method invoked, the Profiler records the time of invocation, and when the local method terminates, it computes the mean time of invocation and the mean number of invocations of each other method during one invocation of the local method. When a local method invokes a remote method, the Profilers compute the mean time required for the remote method invocation. The Profilers operate on a timescale of seconds, and use the above information to construct a local profile of the method invocations.

The Resource Manager maintains a global view of the system and does not need to act individually upon each measurement made by each Profiler. Periodically, the Profilers generate a report to the Resource Manager, which constructs a profile for the entire system. Based on the Profilers' reports, the Resource Manager estimates a system-wide mean time required to invoke each method both locally and remotely and the mean number of invocations of other methods made by this method. The Profilers also compute the current load on the processor's resources (processing and memory) and periodically transmit this information to the Resource Manager. The Resource Manager then decides which is the most appropriate processor to host a new object replica or detects an overloaded processor and uses the migration mechanisms to reallocate objects to different processors.

1.8.1 Dynamic Real-Time Scheduling

The Eternal system uses the least laxity scheduling dynamic real-time scheduling algorithm. In a multi-processor environment, least laxity scheduling has proven to be quite effective since it allows the invocation of a method of a task to take the task's laxity with it, from one processor to another, yielding a system-wide scheduling strategy that requires only local computation. In least laxity scheduling, the laxity of a task represents a measure of urgency of the task, and is defined as:

$$Laxity = Deadline - ComputationTime$$

To determine the real-time priority of the application objects, the laxity value of the objects is augmented with the importance that these objects have

for the application tasks. Then, the objects are executed according to their real-time priorities. We use the Real-Time (RT) Scheduling class provided by the Solaris operating system which provides some degree of real-time support. Threads in the real-time class have a higher priority than threads belonging to any other class, and they run until they voluntarily surrender the processor. The Scheduler is instrumented to allow threads from the other classes with lower priorities also to be able to operate in the system.

In Eternal, the Scheduler works in cooperation with the Profilers and the Resource Manager. The Scheduler keeps a Ready Queue that determines the local order in which the objects are dispatched in the processor. The order is determined by the laxity value of the task invoking the object and the importance the object has for the task. When a new task arrives, the Scheduler calculates a target deadline for the task and then subtracts the task's estimated computation time to yield the initial laxity for the task. The LocateRequest and LocateReply messages sent by a client object are used to identify the initiation of a new task. A LocateRequest message is sent by a client object to obtain the current addressing information for a server object. A LocateReply is a reply message sent by the server object in response to the received LocateRequest message.

However, the actual scheduling begins on receipt of a Request message. A Request message is sent by the client object to invoke an operation of the server object. As the server object executes, it is scheduled according to the task's laxity. The laxity value is adjusted depending on other server objects located on the same processor. The object with the minimum laxity value is scheduled first. If a server object invokes a method of another server object, the invocation message carries the task's laxity with it. That laxity is used to schedule the method on that processor.

When the task completes, a Reply message is sent to the client object. Upon receipt of the Reply message, the Scheduler calculates the Residual Laxity of the task. The Residual Laxity gives us a good estimate of the current system conditions. If the task execution is completed more quickly than was expected, the task laxity increases; otherwise, the task laxity is reduced. If the Resource Manager estimate of the task's computation time is correct, the Residual Laxity is the same as the Initial Laxity.

1.8.2 Object Migration

Object migration involves policies that determine how migration is used and mechanisms for the actual transfer. Resource management in Eternal in-

volves two migration algorithms that decide to migrate objects when the load on a processor is too high or when the latency of a task is too high. In Eternal, the Profilers measure the load on the processors and monitor the behavior of the objects and periodically report this information to the Resource Manager. The Resource Manager in an effort to keep a uniform load on all of the processors, uses the Cooling algorithm to migrate objects from an overloaded processor. It selects the objects that are contributing the most to the load on the overloaded processor and tries to migrate them to the least-loaded processor. The Resource Manager uses the Hotspot algorithm when a task is not meeting its deadlines. It selects the object whose methods cause the larget increase in the latency to the completion of the task and tries to migrate it in the least loaded processor.

To migrate an object, a new object (identical to the one to be migrated) is created on another machine, and the state of the old object is transferred to the new object on that machine. Then, the old object is destroyed and the new object takes over. Migrating an object involves: (1) transfer of the virtual memory of the object, (2) access to any open files using their file descriptors, and, (3) transfer of object-specific information such as the object's identifier, the user's id and the current working directory.

The costs of migration depends on both the policies and the mechanism to be used. The cost of the mechanisms consists of the cost to transfer open files and the speed of transferring virtual memory. Our recent results show that the cost of the policies depends on the frequency at which the Profilers provide feedback information to the Resource Manager. The more frequent the Profiler reports information, the more accurate the processor load is represented by the Resource Manager, which decides the object migration. If this information is sent very frequently, the Resource Manager may decide to migrate objects more often than is required. The application tasks themselves determine how frequently the Profilers report to the Resource Manager.

1.9 The Aroma System

The Java platform provides two distinct models for distributed computing, namely JavaIDL and JavaRMI. The JavaIDL model is a concrete Java implementation of the CORBA specification, and promotes interoperability with other CORBA-compliant objects. The basic JavaRMI model (RMI-JRMP) is targeted at pure Java client-server applications, and provides easy-to-use interfaces with simple semantics. RMI-JRMP exploits the Java Remote Method Protocol (JRMP), a TCP/IP-based protocol that leverages Java-

specific mechanisms such as object serialization, distributed garbage collection and dynamic class-loading.

To support the integration of legacy code with JavaRMI applications, the basic JRMI model was extended to support CORBA's IIOP protocol. RMI-IIOP bridges the JavaRMI and CORBA models, preserving the simple interfaces of RMI-JRMP, exploiting the CORBA2.3 pass-by-value semantics in lieu of object serialization, and doing away with Java-isms such as distributed garbage collection and dynamic class-loading. The RMI-IIOP model is the communication model preferred in the Enterprise Java Beans (EJB 1.1) specification.

The Aroma System is middleware that extends the JRMI model with support for replication. Thus, Aroma can be exploited by distributed Java applications to provide reliable, highly-available operation. Aroma has three objectives:

- **Transparency**. The Aroma mechanisms should be completely invisible to the application. There is no need for the application developer to use special constructs or modify the application in any way, to exploit the replication mechanisms. As a result, even existing applications can take advantage of the Aroma infrastructure for replication.
- **Strong Replica Consistency**. By definition, replicas must be indistinguishable from each other both in internal state as well as in exhibited behavior. Aroma supports replication of both stateful and stateless objects. Thus, to ensure "correct" replicated behavior, Aroma provides strong replica consistency, even in the presence of faults.
- **Flexibility**. Typical applications involve upwards of 5000 objects distributed over a limited number of processors. Aroma support both passive and active replication styles, allowing the system administrator the flexibility of allocating available resources in an optimal manner.

An overview of the Aroma System architecture is provided in Figure 4, and consists of three main components, the Aroma Interceptor, The Aroma Replication Manager and a reliable, totally-ordered multicast protocol, namely Totem. The reliable, ordered delivery guarantees are exploited to enforce replica consistency, and to facilitate communication within and across replica groups.

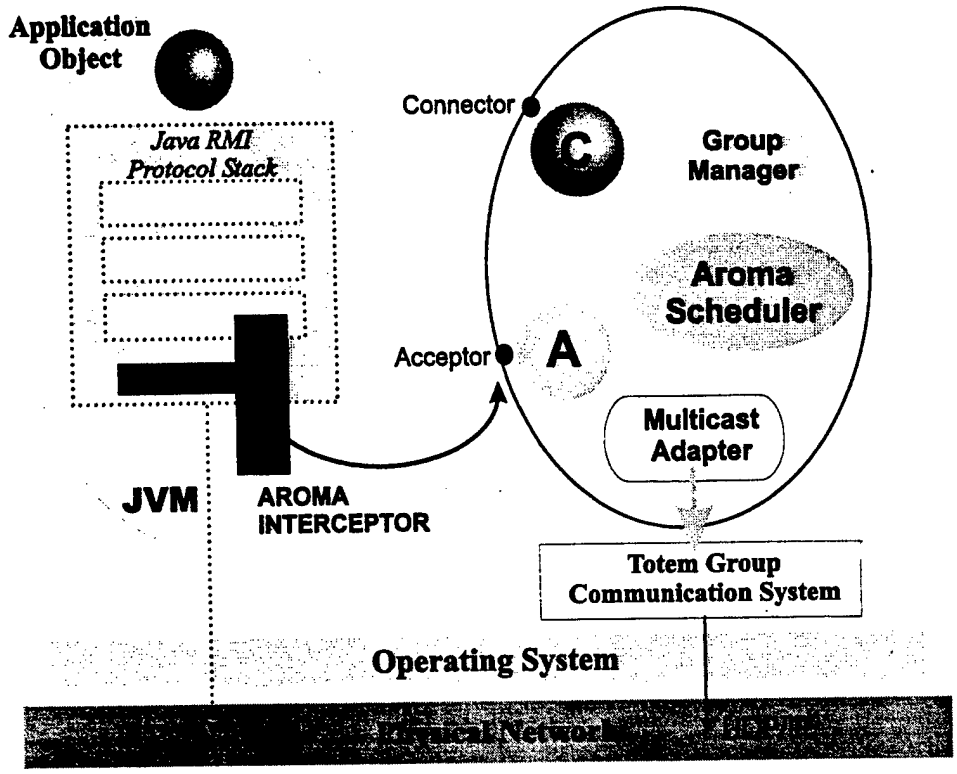


Figure 4: The Aroma system.

1.9.1 Interceptor

The Aroma Interceptor consists of a custom copy of the Java `libnet` networking library, and a `Delegate` object. The *libnet* library mimics the interface of the standard *libnet* Java networking library, and internally forwards the calls to an instance of a `Delegate` class. Aroma provides a base `Delegate` object and a derived `AromaDelegate` object. The base `Delegate` object delegates the calls to the standard *libnet* library, thereby supporting standard JRMI behavior. In this mode, we can exploit the Interceptor for parsing standard JRMI messages, debugging, or for gathering statistcal information about the frequency and content of JRMI invocations. However, by using the `AromaDelegate` class, we map intercepted socket calls onto a "connection interface"; this interface translates calls meant for a TCP/IP socket, into calls made on a local socket to the Aroma Replication Manager. Subsequently, all intercepted TCP/IP messages are diverted, via this connection, to the Aroma replication mechanisms.

Our Interceptor design provides a double advantage. It mimics a standard

interface provided with the JVM; thus, the methods that it implements are guaranteed to be supported by all operating systems that support the Java Virtual Machine. Furthermore, the Interceptor is introduced to the JVM at runtime. Because it requires no modification or addition to the application, it can be exploited both by existing applications and by applications under development. Experiments have shown that the Interceptor functions successfully with both RMI-JRMP and RMI-IIOP models and adds minimal overhead to the system.

1.9.2 Replication Manager

The Aroma Replication Manager is a daemon that runs outside the Java Virtual Machine, and is the cornerstone of the replication process. The Replication Manager defines five distinct components: Acceptor, Connector, Scheduler, Group Manager, and Multicast Adapter. The connector and scheduler components comprise the Aroma Message Handler, and implement the core replication mechanisms.

- **Acceptor.** The Acceptor listens on the well-known port associated with the Aroma daemon, accepts incoming requests, determines the identity of the requestor (group identity), and spawns off an appropriate Connector instance to handle further requests from that client JVM.
- **Connector.** The Acceptor establishes a dedicated Connector for every new replica detected on the local host. The Connector controls the channel between the JVM and the physical network, separating replica-specific information from the details associated with the group. The Connector maintains the mapping between a replica identifier, and its associated group identifier. Network-bound messages are "patched" to contain only group-level identifiers, and are adapted for multicast to all replicas of the target object; the changes are reversed on incoming messages bound for the JVM. The Connector encapsulates the replica consistency mechanisms required to overcome non-determinism in the JRMI architecture.
- **Scheduler.** The Scheduler supervises message queueing at the multicast adapter and the connectors, routing messages between them as required. All network bound messages are collected from the connectors and enqueued at the adapter for multicast. Inbound messages from the network are dequeued from the adapter and routed to the appropriate connectors for delivery to the corresponding replicas. The Scheduler detects and discards duplicate messages. It can be extended

to undertake the logging of invocations and responses required for recovery. The Scheduler exploits the services of the Group Manager to discard all incoming messages destined for groups that do not host a replica on the local processor.

- **Group Manager.** The Group Managers on the distributed Aroma Replication Managers systemwide, collaborate to maintain information about the groups supported by the system, and the membership of each group. All "membership" messages are delivered, by the Scheduler, to the local Group Manager. The Group Manager also identifies groups that host a replica on the local host, and the Connector associated with it.
- **Multicast Adapter.** The Multicast Adapter defines simple *open, send, poll, receive, close* semantics, that are implemented by a concrete multicast protocol, in our case, Totem. By mapping our requests onto this generic `MulticastAdapter` interface, we can exploit any reliable, totally-ordered multicast protocol that can support this interface.

Currently, we have implemented different components of the Aroma System and are working on integration of these mechanisms. The Aroma System has been designed to handle both RMI-JRMP and RMI-IIOP; the prototype implementation is being tested with RMI-JRMP. Aroma requires a Java2 Standard Edition (J2SE) development kit, although it can easily be modified to support earlier versions. Our primary development platform is Solaris 2.6; the code has also been tested successfully on Linux (Mandrake 6.0, 2.2.9 kernel) using the Blackdown-Sun Microsystems port of the J2SE. Future directions include identifying JRMP-specific and IIOP-specific requirements during recovery, and implementing mechanisms to meet these requirements.

2 Participants in the Project

2.1 Professors

Louise E. Moser
P. M. Melliar-Smith

2.2 Ph.D. Students

Vasiliki Kalogeraki
Ruppert Koch
Nitya Narasimhan
Lauren Tewksbury
Wenbing Zhao
Kim Kihlstrom, graduated, August 1999
Priya Narasimhan, graduated, December 1999

2.3 M.S. Students

Fabrice Ferval, graduated, June 1999
Tormod Haavi, graduated, June 1999

3 Publications

Publications 1-5 in the list below are related to the Eternal project but actually preceded the start of the project.

1. Object-Oriented Programming of Complex Fault-Tolerant Real-Time Systems, L. E. Moser, P. Narasimhan and P. M. Melliar-Smith, *Proceedings of the IEEE Second International Workshop on Object-Oriented Real-Time Dependable Systems*, Laguna Beach, CA (February 1996), 116-119.

2. Consistency of Partitionable Object Groups in a CORBA Framework, P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *Proceedings of the 30th Hawaii International Conference on System Sciences*, Maui, HI (January 1997), 120-129.

3. Separation of Concerns: Functionality vs. Quality of Service, P. M. Melliar-Smith, L. E. Moser and P. Narasimhan, *Proceedings of the Third IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, Newport Beach, CA (February 1997), 272-274.

4. Exploiting the Internet Inter-ORB Protocol Interface to Provide CORBA with Fault Tolerance, P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *Proceedings of the 3rd Conference on Object-Oriented Technologies and Systems*, Portland, OR (June 1997), 81-90.

5. The Interception Approach to Reliable Distributed CORBA Objects, P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *Proceedings of the 3rd Conference on Object-Oriented Technologies and Systems*, Portland, OR (June 1997), 245-248.

6. Replica Consistency of CORBA Objects in Partitionable Distributed Systems, P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *Distributed Systems Engineering*, vol. 4 (September 1997), 139-150.

7. Emerging Technologies in Computer Networks and Distributed Systems, K. Berket, R. K. Budhia, K. P. Kihlstrom, R. Koch, N. Narasimhan, P. Narasimhan, E. M. Royer, M. D. Santos, A. Shum, E. Thomopoulos, P. M. Melliar-Smith and L. E. Moser, *IEEE Looking Forward*, vol. 5, no. 3, 2-6.

8. The Eternal System, L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, *Proceedings of the OOPSLA Workshop on Dependable Distributed Object Systems*, Atlanta, GA (October 1997).

9. Soft Real-Time Resource Management in a CORBA Distributed System, V. Kalogeraki, P. M. Melliar-Smith and L. E. Moser *Proceedings of the IEEE*

Workshop on Middleware for Distributed Real-Time Systems and Services (December 1997), San Francisco, CA, 46-51.

10. The Totem Multiple-Ring Ordering and Topology Maintenance Protocol, D. A. Agarwal, L. E. Moser, P. M. Melliar-Smith and R. K. Budhia, *ACM Transactions on Computer Systems* 16, 2 (May 1998), 93-132.

12. Network Protocols, P. M. Melliar-Smith and L. E. Moser, *Computational Grids: The Future of High-Performance Distributed Computing*, ed. I. Foster and C. Kesselman, Morgan-Kaufmann (1998).

13. Group Communication, P. M. Melliar-Smith and L. E. Moser, *Encyclopedia of Electrical and Electronics Engineering*, ed. J. Webster, John Wiley & Sons (February 1999).

14. Replicated Objects, L. E. Moser, *Encyclopedia of Distributed Computing*, ed. J. Urban and P. Dasgupta, Kluwer Academic Publishers (2000).

15. The Totem System, D. A. Agarwal, L. E. Moser and P. M. Melliar-Smith, *Encyclopedia of Distributed Computing*, ed. J. Urban and P. Dasgupta, Kluwer Academic Publishers (2000).

16. Consistent Object Replication in the Eternal System, L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, *Theory and Practice of Object Systems*, vol. 4, no. 2 (1998), 1-12.

17. Supporting Enterprise Applications with the Eternal System, L. E. Moser, P. M. Melliar-Smith, P. Narasimhan, V. Kalogeraki and L. Tewksbury, *Proceedings of the IEEE Conference on Enterprise Networking and Computing '98*, ICC/SUPERCOMM '98, Atlanta, GA (June 1998).

18. Performance Engineering of the Totem Group Communication System, R. K. Budhia, L. E. Moser and P. M. Melliar-Smith, *Distributed Systems Engineering Journal* 5, 2 (June 1998), 78-87.

19. The Realize Middleware for Replication and Resource Management, P. M. Melliar-Smith, L. E. Moser, V. Kalogeraki and P. Narasimhan, *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, Middleware '98*, The Lake District, England (September 1998), 123-138.

20. Fault Tolerance for CORBA, Version 1.0, Initial RFP Submission, OMG Document orbos/98-10-08, L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, Technical Report 98-27, Department of Electrical and Computer Engineering, University of California, Santa Barbara (October 1998).

21. Analyzing and Measuring the Latency of the Totem Multicast Protocols, E. Thomopoulos, L. E. Moser and P. M. Melliar-Smith, *Computer Networks: The International Journal of Computer and Telecommunications Networking* 31, 1-2 (1999), 57-76.

22. Dynamic Modeling of Replicated Objects for Dependable Soft Real-Time Distributed Object Systems, V. Kalogeraki, L. E. Moser and P. M. Melliar-Smith, *Proceedings of the 4th IEEE Workshop on Object-Oriented Real-Time Dependable Systems* Santa Barbara, CA (January 1999).

23. Using Multiple Feedback Loops for Object Profiling, Scheduling and Migration in Soft Real-Time Distributed Object Systems, V. Kalogeraki, P. M. Melliar-Smith and L. E. Moser, *Proceedings of the IEEE 2nd International Symposium on Object-Oriented Real-Time Distributed Computing*, Saint Malo, France (May 1999), 291-300.

24. Providing Support for Survivable CORBA Applications with the Immune System, P. Narasimhan, K. P. Kihlstrom, L. E. Moser and P. M. Melliar-Smith, *Proceedings of the IEEE 19th International Conference on Distributed Computing Systems* Austin, TX (May/June 1999), 507-516.

25. A Fault Tolerance Framework for CORBA, L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, *Proceedings of the IEEE 29th Annual International Symposium on Fault-Tolerant Computing*, Madison, WI (June 1999), 150-157.

26. Using Interceptors to Enhance CORBA, P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *IEEE Computer*, vol. 32, no. 7 (July 1999), 62-68.

27. Replication and Recovery Mechanisms for Strong Replica Consistency in Reliable Distributed Systems, P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *Proceedings of the Fifth ISSAT International Conference on Reliability and Quality Design*, Las Vegas, NV (August 1999), 26-31.

28. Multicast Group Communication for CORBA, L. E. Moser, P. M. Melliar-Smith, P. Narasimhan, R. R. Koch and K. Berket, *Proceedings of the IEEE International Symposium on Distributed Objects and Applications*, Edinburgh, Scotland (September 1999), 98-109.

29. The Eternal System: An Architecture for Enterprise Applications, L. E. Moser, P. M. Melliar-Smith, P. Narasimhan, L. A. Tewksbury and V. Kalogeraki, *Proceedings of the IEEE Third International Enterprise Distributed Object Computing Conference*, Mannheim, Germany (September 1999), 214-222.

30. A Group Communication Protocol for CORBA, L. E. Moser, P. M. Melliar-Smith, R. R. Koch and K. Berket, *Proceedings of the IEEE International Workshop on Group Communication*, Aizu, Japan (September 1999), 30-36.

31. Enforcing Determinism for the Consistent Replication of Multithreaded CORBA Applications, P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *Proceedings of the IEEE 18th Symposium on Reliable Distributed Systems*, Lausanne, Switzerland (October 1999), 263-273.

32. Transparent Fault Tolerance for CORBA, P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *Proceedings of the IEEE Workshop on Reliable Middleware*, Lausanne, Switzerland (October 1999), 7-13.

33. Transparent Fault Tolerance for CORBA, P. Narasimhan, Technical Report 99-18, Department of Electrical and Computer Engineering, University of California, Santa Barbara (December 1999).

34. A CORBA Framework for Managing Real-Time Distributed Multimedia Applications, V. Kalogeraki, P. M. Melliar-Smith and L. E. Moser, *Proceedings of the IEEE 33rd Hawaii International Conference on System Sciences*, Maui, HI (January 2000).

35. Eternal: Fault Tolerance and Live Upgrades for Distributed Object Systems, L. E. Moser, P. M. Melliar-Smith, P. Narasimhan, L. A. Tewksbury and V. Kalogeraki, *Proceedings of the DARPA Information Survivability Conference and Exposition*, Vol. II, Hilton Head, SC (January 2000), 184-196.

36. The SecureGroup Group Communication System, L. E. Moser, P. M. Melliar-Smith and N. Narasimhan, *Proceedings of the DARPA Information Survivability Conference and Exposition*, Vol. II, Hilton Head, SC (January 2000), 256-270.

37. Realize: Resource Management for Soft Real-Time Distributed Systems, P. M. Melliar-Smith, L. E. Moser and V. Kalogeraki, *Proceedings of the DARPA Information Survivability Conference and Exposition*, Vol. I, Hilton Head, SC (January 2000), 281-293.

38. Dynamic Scheduling for Soft Real-Time Distributed Object Systems, V. Kalogeraki, P. M. Melliar-Smith and L. E. Moser *Proceedings of the IEEE 3rd International Symposium on Object-Oriented Real-Time Distributed Computing*, Newport Beach, CA (March 2000), 114-121.

39. Fault Tolerant CORBA, Adopted Specification, OMG Document ptc/2000-

04-04, (April 2000), http://www.omg.org/cgi-bin/doc?ptc/2000-04-04

40. Gateways for Accessing Fault Tolerance Domains P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *Middleware 2000: IFIP/ACM International Conference on Distributed Systems Platforms Lecture Nodes in COmputer Science 1795*, New York, NY (April 2000), 88-10

41. Interception in the Aroma System, N. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *Java Grande Conference*, San Francisco, CA (June 2000), to appear.

42. R. R. Koch, L. E. Moser and P. M. Melliar-Smith A Reliable Many-to-Many Multicast Protocol for Group Communication over ATM Networks, *Proceedings of the IEEE International Conference on Dependable Systems and Networks* New York, NY (June 2000), to appear.

43. Transparent Fault Tolerance for Enterprise Applications, *Proceedings of the SSGRR 2000 Computer and eBusiness Conference*, L'Aquila, Rome, Italy (July/August 2000), to appear.

44. Applying Design Patterns to Build Transparent Fault Tolerance Middleware P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *Theory and Practice of Object Systems*, to appear.

45. Profiling, Scheduling and Migration in Real-Time Distributed Object Systems, V. Kalogeraki, P. M. Melliar-Smith and L. E. Moser, submitted.

46. Recovery of Strongly Consistent Replicated CORBA Objects, P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, submitted.

47. Transparent Consistent Replication of JavaRMI Objects, N. Narasimhan, L. E. Moser, P. M. Melliar-Smith, submitted.

48. Dynamic Scheduling of Distributed Method Invocations, V. Kalogeraki, L. E. Moser, P. M. Melliar-Smith, submitted.

49. The Eternal System, P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, *Encyclopedia of Distributed Computing*, ed. J. Urban and P. Dasgupta, Kluwer Academic Publishers (2000), in preparation.

50. Distributed Object Computing, L. E. Moser and P. M. Melliar-Smith, *Encyclopedia of Distributed Computing*, ed. J. Urban and P. Dasgupta, Kluwer Academic Publishers (2000), in preparation.

4 Other Activities

We list below meetings attended, presentations and software demonstrations, and visitors to the project.

4.1 DARPA Meetings, Presentations, Demos

L. E. Moser and P. M. Melliar-Smith, Demonstration of Totem at the DoD Research Advocacy Day and also at DARPA, Washington, DC (May 1997)

L. E. Moser, Extending CORBA with Fault Tolerance and Real Time, Distributed Objects Session, DARPA PI meeting, Washington, D.C. (July 1997)

L. E. Moser, The Eternal System, DARPA Fault-Tolerant Computing Workshop, Jet Propulsion Laboratory, Pasadena, CA (September 1997)

L. E. Moser, The Eternal System, 20th annual C3A Technical Exchange Meeting (TEM 97), Rome Labs (December 1997)

P. M. Melliar-Smith, The Realize System, DARPA-OMG-MCC Workshop on Compositional Software Architectures, Monterey, CA (January 1998)

L. E. Moser, The Eternal System, DARPA-OMG-MCC Workshop on Compositional Software Architectures, Monterey, CA (January 1998)

L. E. Moser, The Eternal System, DARPA Adaptive Architecture Workshop, SRI International, Menlo Park, CA (May 1998)

P. M. Melliar-Smith, L. E. Moser, R. Koch and M. Santos, Demonstration, The Realize System, Totem System and Atomic Group System, DARPA PI Meeting, San Diego, CA (July 1998)

P. M. Melliar-Smith, L. E. Moser, DARPA PI Quorum Meeting, Poster Presentation on the Realize System, Atlanta, GA (February 1999)

L. E. Moser, P. M. Melliar-Smith, P. Narasimhan, DARPA Information Survivability Conference, Demonstration of the Eternal System, Hilton Head, SC (January 2000)

4.2 Other Meetings, Presentations, Demos

P. Narasimhan, Exploiting the Internet Inter-ORB Protocol to Provide Fault Tolerance for CORBA, 3rd Conference on Object-Oriented Technologies and Systems, Portland, OR (June 1997)

P. Narasimhan, The Interception Approach to Providing CORBA with Fault Tolerance, 3rd Conference on Object-Oriented Technologies and Systems,

Portland, OR (June 1997)

L. E. Moser, The Eternal System, OOPSLA Workshop on Dependable Distributed Object Systems, Atlanta, GA (October 1997)

L. E. Moser, P. M. Melliar-Smith, V. Kalogeraki, P. Narasimhan, N. Narasim han, L. Tewksbury and K. Kihlstrom, Fault-Tolerant Distributed Systems, Jet Propulsion Laboratory (November 1997)

L. E. Moser, The Eternal System, Annual UCSB College of Engineering Research Review (November 1997)

L. E. Moser and P. M. Melliar-Smith, Strategies for Building Fault-Tolerant Distributed Systems, Channel Islands Chapter American Society of Naval Engineers, Pt Mugu (February 1998)

P. M. Melliar-Smith, Supporting Enterprise Applications with the Eternal System, IEEE Conference on Enterprise Networking and Computing '98, ICC/SUPERCOMM '98, Atlanta, GA (June 1998)

P. M. Melliar-Smith, The Realize Middleware for Replication and Resource Management, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, Middleware '98, The Lake District, England (September 1998)

P. M. Melliar-Smith, L. E. Moser, Priya Narasimhan, Vana Kalogeraki, Lauren Tewksbury, Fault Tolerance for CORBA, OMG Meeting, Burlingame, CA (November 1998)

K. Berket, The InterGroup Protocols: Scalable Group Communication for the Internet, Third Global Internet Mini-Conference, GLOBECOM, Sydney, Australia (November 1998)

R. R. Koch, Global Causal Ordering with Minimal Latency, Conference on Parallel and Distributed Computing and Networks, Brisbane, Australia (December 1998)

R. R. Koch, Timestamp Acknowledgments for Determining Message Stability, Conference on Parallel and Distributed Computing and Networks, Brisbane, Australia (December 1998)

V. Kalogeraki, Dynamic Modeling of Replicated Objects for Soft Real-Time Distributed Systems, IEEE 4th Workshop on Object-Oriented Real-Time Dependable Systems, Santa Barbara, CA (January 1999)

P. M. Melliar-Smith, L. E. Moser, Priya Narasimhan, OMG Meeting, Washington, DC (January 1999)

P. M. Melliar-Smith, L. E. Moser, P. Narasimhan, OMG Meeting, Philadel-

phia, PA (March 1999)

V. Kalogeraki, Using Multiple Feedback Loops for Object Profiling, Scheduling and Migration in Soft Real-Time Distributed Object Systems, IEEE 2nd International Symposium on Object-Oriented Real-Time Distributed Computing, Saint Malo, France (May 1999)

P. Narasimhan, Providing Support for Survivable CORBA Applications with the Immune System, IEEE 19th International Conference on Distributed Computing Systems, Austin, TX (May/June 1999)

L. E. Moser, P. M. Melliar-Smith, Priya Narasimhan, Vana Kalogeraki, Lauren Tewksbury, Ruppert Koch, Nitya Narasimhan, Wenbing Zhao, OMG Fault Tolerance for CORBA Submitters Meeting, Santa Barbara, CA (June 1999)

L. E. Moser, A Fault Tolerance Framework for CORBA, IEEE 29th Annual International Symposium on Fault-Tolerant Computing, Madison, WI (June 1999)

L. E. Moser, P. M. Melliar-Smith, Priya Narasimhan, OMG Meeting, San Jose, CA (August 1999)

P. Narasimhan, Replication and Recovery Mechanisms for Strong Replica Consistency in Reliable Distributed Systems, Fifth ISSAT International Conference on Reliability and Quality Design, Las Vegas, NV (August 1999)

L. E. Moser, Multicast Group Communication for CORBA, IEEE International Symposium on Distributed Objects and Applications, Edinburgh, Scotland (September 1999)

P. M. Melliar-Smith, The Eternal System, Iona Technologies, Dublin, Ireland (September 1999)

R. R. Koch, A Group Communication Protocol for CORBA, IEEE International Workshop on Group Communication, Aizu, Japan (September 1999)

P. M. Melliar-Smith, The Eternal System: An Architecture for Enterprise Applications, IEEE Third International Enterprise Distributed Object Computing Conference, Mannheim, Germany (September 1999)

P. Narasimhan, Enforcing Determinism for the Consistent Replication of Multithreaded CORBA Applications, IEEE 18th Symposium on Reliable Distributed Systems, Lausanne, Switzerland (October 1999)

P. Narasimhan, Transparent Fault Tolerance for CORBA, IEEE Workshop

on Reliable Middleware, Lausanne, Switzerland (October 1999)

P. M. Melliar-Smith, Fault Tolerant CORBA, OMG Meeting, Cambridge, MA (November 1999)

L. E. Moser, The Eternal System, Telcordia, Morristown, NJ (November 1999)

P. M. Melliar-Smith, Fault Tolerant CORBA, OMG Meeting, Mesa, AZ (January 2000)

L. E. Moser, P. M. Melliar-Smith, P. Narasimhan, R. Koch, W. Zhao, Demonstration of the Eternal System, Air Defense, OMG Meeting, Mesa, AZ (January 2000)

V. Kalogeraki, A CORBA Framework for Managing Real-Time Distributed Multimedia Applications, IEEE 33rd Hawaii International Conference on System Sciences, Maui, HI (January 2000)

L. E. Moser, P. M. Melliar-Smith, P. Narasimhan, Demonstration of the Eternal System, Air Defense, OMG Meeting, Denver, CO (March 2000)

V. Kalogeraki, Dynamic Scheduling for Soft Real-Time Distributed Object Systems, IEEE 3rd International Symposium on Object-Oriented Real-Time Distributed Computing, Newport Beach, CA (March 2000)

P. Narasimhan, Gateways for Accessing Fault Tolerance Domains, Middleware 2000: IFIP/ACM International Conference on Distributed Systems Platforms New York, NY (April 2000)

L. E. Moser, P. M. Melliar-Smith and their students also made presentations and gave software demonstrations for the following visitors to the project.

4.3 Visitors to the Project

Professor Klaus Petermann, Technical University of Berlin
Professor Ben Wah, University of Illinois
Dr. Kevin C. Almeroth, Georgia Institute of Technology
Dr. Ender Ayanoglu, Bell labs, Lucent Technologies
Dr. Rachid Guerraoui, Ecole Polytechnique Federale de Lausanne
Dr. David Blumenthal, Georgia Institute of Technology
Richard Thibault and Bruce Canna, The Foxboro Company, Foxboro, MA
Professor Dan Gajski, University of California, Irvine
Professor Douglas Schmidt, Washington University, St. Louis
Dr. Michael Reiter, AT&T Laboratories

Dr. Gil Neiger, Intel
Dr. Brian A. Hanson, Director, Panasonic Technologies, Inc
with 12 other Directors of Panasonic's USA Laboratories
Keith Bromley, NRad, San Diego
Dennis Hollingworth, Trusted Information Systems
C. K. Toh, Hughes Research Laboratory
Brian Norling, Director of Engineering, Space and Launch Systems,
Litton Guidance and Control
Professor Partha Dasgupta, Arizona State University
Professor Hermann Kopetz, Technical University of Vienna, Austria
James Kirkley, John Norris and Brian Whittle, QAD, Carpenteria, CA
Dr. David Lomet, Microsoft Corporation
Dr. Lewis B. Oberlander, Dr. Won Kang, Francis Tam,
Motorola Corporation, Arlington Heights, IL
Dr. Gregory Papodopolous, Dr. Emil Sarpa, Dr. John M. Hale,
Georgi C. Johnson, Sun Microsystems
Professor Douglas Schmidt, Washington University, St. Louis
Dr. Stephen Wright, Mathematics and Computer Science Division,
Argonne National Laboratory
Professor Georgis Giannakis, University of Virginia
Gerhard Beenan, Greg Hoffman, Mihir Ravel, Tektronix
Mark Gibbs and John Dix, Network World
Mike Toma and Vic Walker, jeTech Data Systems, Inc, Camarillo, CA
Professor Odd Pettersen, Norwegian University of Science and Technology
Professor Robert Blumofe, University of Texas, Austin
Dr. Peter Feldmann, Bell Labs
Professor Yale Patt, University of Michigan
Professor David Du, University of Minnesota
Achilleas Anastasapoulos, University of Southern California
Professor Upamanyu Madhow, University of Illinois
Dr. Ragunathan Rajkumar, Carnegie Mellon University
Ali Dasdan, University of Illinois
Professor Dian Zhou, University of North Carolina
Bwolen Yang, Carnegie Mellon University
Edward Chang, Stanford University
Professor Peder Emstad, Norwegian University of Science and Technology
Tom Bruggere, Mentor Graphics
Pascal Felber, Oracle Corporation
Victor Giddings, Objective Interfaces
Chris Smith, Ericsson

Shalini Yajnik, Lucent Technologies
Patrick Hurley, Vaughn Combs, Program Managers from
Air Force Laboratory Rome
Dr. Thomas McVittie, Joe Hutcherson, Mark Jean, NASA JPL
Dr. Shahzad Aslam-Mir, Indresh Chaudhari, Joey Garon, Yuval Levy, Vertel
Richard Santagelo, Guy Savage, QAD
Frank Careccia, Punkaj Jain, Anne Greenlee, Erik O'Neill, Inprise
Benn Schreiber, Santa Barbara Technology Incubator

4.4 Industrial Interest

The Totem system has attracted considerable interest from individuals in a number of organizations.

Joe Caruso, Leslie Madden and Michael Masters of the HiPerD project at NSWC, Dalgren, have expressed interest in using the Eternal and Realize systems for the SC21 project. Quite extensive discussions have been conducted with Joe Caruso on our experience with CORBA and on the use of CORBA, Eternal and Realize in the next generation projects at NSWC.

Keith Bromley of NRaD visited our Lab in October 1997. Keith had heard Louise Moser's presentation at the DARPA meeting in Washington, D.C. in July 1997, and had recommended that we present our work at the Jet Propulsion Laboratory (JPL) to Leon Alkalai, Roger Lee and the other researchers at JPL involved in the next generation space program X2000. Louise Moser made a presentation in September 1997 at the DARPA sponsored workshop on fault tolerance at JPL. Louise Moser and Michael Melliar-Smith and five of their students made a second presentation at JPL in November 1997. JPL is very interested in using the technology being developed by Louise Moser and Michael Melliar-Smith and their students at UCSB, both in this project and in our other DARPA sponsored projects.

Patrick Hurley, our project manager at Rome Labs for the Eternal system project, is also interested in using this technology.

Dr. Deborah Agarwal, our former student and now a research scientist at Lawrence Berkeley National Laboratory, is currently collaborating with us to develop protocols for DOE's Distributed Collaboration Environments to allow scientists to collaborate over the Internet. Two of sour students, Karlo Berket and Nitya Narasimhan, spent the summer of 1997 at LBNL.

Dr. Hossein Moiin, our former student and an employee of Sun Microsys-

tems continues to seek our advice and to provide equipment to us. At Sun, we have given talks on Totem, Eternal and Realize, presented the air traffic control demonstration, and have met with people in Sun's STREAMS, NEO, and High Availability groups.

Gradimir Starovic and Martin Mayhead of SUN IMP in England, who build fault-tolerant real-time systems using Sun processors, are also interested in using Totem in telco applications. Hossein Moiin has recently moved to London to work with Sun IMP. The SUN IMP group has partnered with us on our Proposal to the OMG on Fault Tolerance for CORBA.

Michel Gien, Michel Tombroff and Stephane Ciceron of Chorus Systems, recently purchased by Sun, have expressed interest in Eternal and Realize for use with their COOL ORB. Sun Microsystems has purchased Chorus Systems to strengthen Sun's real-time capability.

In December 1997 we presented work on Eternal and Realize at Sun Microsystems in Menlo Park, and Greg Papodopolous, the Chief Technical Officer of Sun Microsystems, and some of his colleagues visited our laboratory in March 1998. In his subsequent debriefing with the Dean of Engineering at UCSB, Dr. Papodopolous commented on the remarkably close correspondence between the research being undertaken by us and the research needs of Sun.

Richard Thibault and Bruce Canna of The Foxboro Company, Foxboro, Massachusetts, manufacturers of industrial control systems are currently starting to develop an object-oriented infrastructure for their control system. Initially, they wanted to use Totem to support a replicated name server. During the visit, however, it became clear that the Eternal and Realize projects correspond more closely to their needs.

David McKnight, Corey Minyard and Gregory Graham of Nortel are implementing a version of Totem in Nortel's ATM telephone switching network to operate across the United States. Two of our students, Ruppert Koch and Efstratios Thomopoulos, spent the summer of 1997 at Nortel as interns working on that implementation.

Dr. Richard Chenovick of Raytheon Electromagnetic Systems Division (now a subsidiary of E-Systems) has visited our laboratory to discuss our technology for various applications.

Brian Norling of Litton Guidance and Control, Space and Launch Systems, visited our laboratory in November 1997. Litton is interested in building commercial low-earth-orbit satellites using essentially the same strategies and technologies envisaged by JPL for deep-space missions.

Lewis Oberlander, Francis Tam and Won Kang of Motorola's Cellular Infrastructure Group at Arlington Heights, Illinois visited the project in January 1998. They are building a new infrastructure for their land-based and land-mobile products, but are not involved in the space-based systems such as Iridium. A part of their plans involves extensive use of object-oriented technology and a real-time ORB, possibly the Sun/Chorus COOL ORB or Doug Schmidt's TAO ORB.

Much of what we described to the group from Motorola about Eternal and Realize exactly matched their needs, and our overheads are acceptable for their application. They were very concerned, and rightly, about our testing procedures and the consequent quality of our code for their telco service-critical applications. They were also concerned that our timescale is still too long for their needs; they wanted a prototype in 3Q98 and a commercial quality product in 2Q99. We have advised them that we might have something more experimental than a prototype in 3Q98 and that there is no way, even if Motorola gives us an army of programmers, that a product could reach telco service-critical quality by 2Q99.

However, it is our belief that Motorola, like NASA JPL, NSWC, Rome Labs, Nortel and Foxboro before them, have nowhere else to go to get transparent object replication, fault tolerance and live upgrades, and they have not denied that. We plan to continue to work with Motorola and to have them experiment with and evaluate our Eternal and Realize systems as soon as it becomes possible to provide useful functionality to them.

Following the presentation of our proposal on Fault Tolerance for CORBA at the OMG meeting at Burlingame, CA, in November 1998, we were approached by several commercial ORB vendors. Major players in the enterprise software market who approached us were Dave Curtis and Jeff Mischinsky of Inprise, Martin Chapman of Iona, and Gary Hallmark of Oracle. We were also approached by smaller specialist real-time ORB vendors, including Tom Greene of Expersoft/Vertel, Ken Black and Jon Currey of Highlander Communications and Bill Beckwith and Victor Giddings of Objective Interface Systems. Our proposal was submitted jointly with Sun Microsystems with whom we are, of course, also in contact.

An appropriate strategy for real-time applications will be to integrate our technology into the ORB, thus avoiding conflicts between multiple levels of scheduling. Specialized real-time ORB vendors are, however, small companies with limited development and marketing resources. They are very interested in integrating our technology into their ORBs but are unlikely to be able to do so without strong customer encouragement. We will work with

military agencies, such as Rome Labs and NSWC, and also with Motorola and Foxbro, to provide that encouragement.

In contrast, the enterprise market, driven by the Internet, has much shorter development cycles and a substantial demand for fault tolerance is developing quite quickly. Few enterprise applications require real-time deadline scheduling, but many of them can benefit from distributed resource allocation and from monitoring and profiling. The design of the Eternal system, with its emphasis on application transparency and operation with existing unmodified commercial ORBs, makes it particularly attractive to the enterprise market, and also to ORB vendors seeking to exploit this market rapidly.

Several of the ORB vendors expressed great interest in the use of our Eternal and Realize technology with their ORBs. We were invited to present and demonstrate the technology to them. We have visited Inprise Corporation and Objective Interface Systems in January 1999, Iona in August 1999 and Vertel in November 1999. We are investigating with these ORB vendors the possibility of licensing our products as augmentations to their ORBs, or of licensing our code and technology for integration into their ORBs.

We worked with these ORB vendors, and several other companies including Lucent Bell Laboratories, Oracle, Lockheed Martin, Ericsson and Sun Microsystems, to develop a common proposal for the standard for fault tolerance in CORBA. The specification provides the stringent fault tolerance that is needed for major government projects, and is based on the DARPA-funded research of this project. The OMG standard for Fault Tolerant CORBA was approved in March 2000, and commercial products should become available during the Fall of 2000.

Considerable interest is also starting to develop in fault tolerance for Java programs, particularly from Sun Microsystems and Oracle. Although the standardization process has not yet started for Java, much of the CORBA fault tolerance technology is directly transferable to Java, and we expect a commercial Java product to be available on a similar timescale.

In October 1999, because of our leadership in the development of the Fault Tolerant CORBA standard, we were approached by Dr. Thomas McVittie and his team at NASA JPL. They sought fault tolerance technology for their Shared Net project being built for the US Marine Corps. The JPL team has visited Santa Barbara twice and we have negotiated a contract with them that will lead to on-ship operation in early 2001.

The Eternal system has been demonstrated at the OMG meeting in Mesa, AZ, in January 2000, the DARPA Information Survivability Conference in

Hilton Head, SC, in January 2000, and the OMG meeting in Denver, CO, in March 2000. As a result of these demonstrations, more than 40 companies have approached us and requested evaluation copies of the Eternal system. Among those companies were Alcatel, AT&T, Boeing, Deutsche Telekom, General Dynamics, GTE BBN, Hitachi, IO Software, Litton, Lockheed Martin, Lucent Technologies, Motorola, NTT, Naval Surface Warfare Center (Aegis), Oracle, Siemens, Sprint, Sun Microsystems, Thompson/CSF and Xerox.

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of Information Systems Science and Technology to meet Air Force unique requirements for Information Dominance and its transition to aerospace systems to meet Air Force needs.