The Eternal System

L. E. Moser, P. M. Melliar-Smith
P. Narasimhan, V. Kalogeraki, L. Tewksbury

Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106

Abstract

The Eternal system replicates CORBA objects to provide fault tolerance and evolution. By exploiting the IIOP interface, Eternal renders the replication transparent not only to the application objects but also to the ORB and thus works with standard commercial CORBA ORBs. The difficult issues of replication, consistency, fault detection and recovery are handled by Eternal and are hidden from the application programmer.

1 Introduction

The Common Object Request Broker Architecture (CORBA) [10, 12] established by the Object Management Group (OMG) facilitates the development of distributed applications by providing for location transparency, separation between specification and implementation, and interworking of heterogeneous languages, architectures and operating systems.

The Eternal system supports the development of distributed object applications by extending CORBA with capabilities for object replication, fault tolerance and evolution of live objects. Eternal operates on top of standard operating systems and works with standard commercial CORBA Object Request Brokers (ORBs). Eternal replicates objects, maintains the consistency of replicated objects and distributes the objects across the system. Both client and server objects can be replicated, and objects can act as both clients and servers. The degree of replication, the type of replication and the location of the replicas are hidden by the object group abstraction. An object group is simply a collection of objects, and the replicas of an object are members of the same object group.

In Eternal, an application programmer programs a CORBA distributed object application in the conventional manner, exploiting the portability, location transparency and interoperability that CORBA provides. Eternal renders that conventional application fault-tolerant by replicating objects. The application programmer does not need to address the difficult issues of replication, consistency, fault detection and recovery that are inherent in such applications. These issues are handled by Eternal and are transparent to the application programmer.

The Eternal system provides

- Support for the programming of fault-tolerant distributed object applications
- Separation of the functional and resource management aspects of the application
- Robust and consistent replication of objects across the distributed system
- Automatic upgrade and evolution of objects while the system is live.

This research has been supported by DARPA and AFOSR, Contracts N00174-95-K-0083 and 442530-25374, and by Sun Microsystems and Rockwell Science Center through the State of California MICRO Program, Grants 96-051 and 96-052.

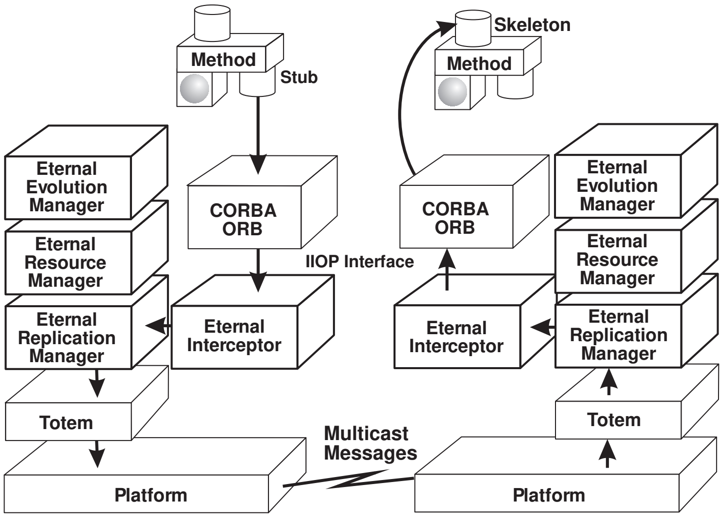

Figure 1: The structure of the Eternal system, consisting of the Interceptor, Replication Manager, Resource Manager and Evolution Manager.

Other systems for developing distributed object applications include OpenDREAMS [2], Electra [4, 5], Orbix+Isis [4] and Arjuna [11]. All of these systems provide some form of fault tolerance but, unlike Eternal, they do not provide for live upgrade and evolution of the objects.

2 Structure of the Eternal System

The structure of the Eternal system is shown in Figure 1. The application objects operate over standard commercial implementations of CORBA and, currently, over standard System V implementations of Unix. Neither the CORBA ORB nor the operating system need to be modified in the *interception approach* [8] that Eternal employs.

The CORBA ORB packages the operations to be performed on the objects into messages, formatted according to the Internet Inter-Orb Protocol (IIOP), which is layered on top of TCP/IP, for communication to objects hosted by the same or different ORBs. Each CORBA object, on its creation, is assigned a unique Unix process identifier or *pid*. The operating system calls made by the object can be monitored [1] using the standard */proc* interface between the object and Unix.

The Interceptor exploits the */proc* interface to monitor the operating system calls made by an object to establish an IIOP connection over TCP/IP, and to communicate IIOP messages over that connection. The Interceptor, however, intercepts the IIOP messages before they reach TCP/IP, and diverts them to the Replication Manager. The Replication Manager then passes the messages to the Totem group communication system [6], which multicasts the messages to the object groups containing the replicas. Any group communication system that provides guarantees similar to those of Totem can be used instead.

The Replication Manager transparently translates invocations of operations on an object group into invocations of operations on the individual member replicas. It provides fault detection and recovery, and maintains consistency of the states of the object replicas. To achieve replica consistency, the Replication Manager utilizes Totem's reliable totally ordered message delivery. In addition, the Replication Manager detects duplicate invocations and duplicate responses, transfers state between replicas, schedules concurrent operations consistently, and provides fulfillment operations to restore a consistent state after network partitioning and remerging. More details about these mechanisms can be found in [7, 9].

The Resource Manager [3], using Interface Definition Language (IDL) object definitions in the Interface Repository, determines the number of object replicas to create and distributes the replicas across the system. To determine appropriate allocations of objects to processors, it measures task behavior to derive profiles of application tasks and objects, and monitors the use of resources such as the load on the processors. Three levels of feedback are applied. On millesecond timescales, measurements of execution times are used to adjust the remaining task laxity for soft real-time scheduling. On subsecond timescales, measurements are used to adjust the method and task profiles from which laxities are estimated. On multiple second timescales, task latencies and loads on resources are used to reassign objects to processors, moving objects when necessary to balance loads or reduce task latencies.

The Evolution Manager performs the automated upgrade and evolution of objects while the system is live, using the IDL definitions retrieved from the Interface Repository. The replication of objects allows the replicas to be modified one at a time while operations continue with the other replicas. The Evolution Manager constructs a sequence of upgrades of individual replicas, adding upgraded methods or removing old methods, to achieve the desired overall upgrade. It uses the Resource Manager to create the replicas of the new versions of the objects and to remove the replicas of the old versions.

The Evolution Manager and the Resource Manager are themselves implemented as collections of CORBA objects, and can thereby benefit from CORBA's interoperability and Eternal's fault tolerance and live upgrade capabilities.

3 Conclusion

As distributed object applications become larger and more complex, the need for fault tolerance will become ever greater. To provide fault tolerance, the Eternal Replication Manager replicates objects and maintains the consistency of the states of the object replicas. The Eternal Interceptor renders this replication transparent to the application objects without modification of the ORB or of the operating system. As the complexity of real-time distributed applications continues to increase, dynamic resource managers that exploit profiling and feedback, as in the Eternal Resource Manager, will become increasingly important. Complex applications are continuously evolving and changing, and the live upgrade capabilities provided by the Eternal Evolution Manager will also become important.

References

[1] A. D. Alexandrov, M. Ibel, K. E. Schauser and C. J. Scheiman, "Extending the operating system at the user level: The Ufo global file system," *Proceedings of the USENIX 1997 Annual Technical Conference*, Anaheim, CA (January 1997), pp. 77-90.

[2] P. Felber, B. Garbinato and R. Guerraoui, "Designing a CORBA group communication service," *Proceedings of the IEEE 15th Symposium on Reliable Distributed Systems*, Niagara on the Lake, Canada (October 1996), pp. 150-159.

[3] V. Kalogeraki, P. M. Melliar-Smith and L. E. Moser, "Soft real-time resource management in CORBA distributed systems," to appear in *Proceedings of the IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, San Francisco, CA (December 1997).

[4] S. Landis and S. Maffeis, "Building reliable distributed systems with CORBA," *Theory and Practice of Object Systems*, vol. 3, no. 1, John Wiley & Sons Publishers, New York (April 1997).

[5] S. Maffeis, "Electra — Making distributed programs object-oriented," *Proceedings of the USENIX Symposium on Experiences with Distributed and Multiprocessor Systems* IV, San Diego, CA (September 1993), pp. 143-156.

[6] 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*, vol. 39, no. 4 (April 1996), pp. 54-63.

[7] L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, "Consistent object replication in Eternal," submitted.

[8] P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, "Exploiting the Internet Inter-ORB protocol interface to provide CORBA with fault tolerance," *Proceedings of the Third USENIX Conference on Object-Oriented Technologies and Systems*, Portland, OR (June 1997), pp. 81-90.

[9] P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, "Replica consistency of CORBA objects in partitionable distributed systems," *Distributed Systems Engineering*, vol. 4, no. 3 (September 1997), pp. 139-150.

[10] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.1, OMG Technical Document PTC/97-09-01, Object Management Group (August 1997).

[11] G. D. Parrington, S. K. Shrivastava, S. M. Wheater and M. C. Little, "The design and implementation of Arjuna," *USENIX Computing Systems Journal*, vol. 8, no. 3 (Summer 1995), pp. 255-308.

[12] R. M. Soley, *Object Management Architecture Guide*, Object Management Group, OMG Document 92-11-1.