

Separation of Concerns: Functionality vs. Quality of Service

P. M. Melliar-Smith, L. E. Moser and P. Narasimhan
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
pmms@ece.ucsb.edu, moser@ece.ucsb.edu, priya@alpha.ece.ucsb.edu

Abstract

The development of complex future applications will depend on separation of concerns between the functional behavior of the application, probably represented by an object-oriented program, and the resource management of the application. In particular, effective fault tolerance will depend on hiding the replication of objects and their distribution to multiple processors.

1 Introduction

We come to the object-oriented world from a background in fault-tolerant distributed systems. Traditionally, such systems have been difficult to program and have required programmers skilled both in the application domain and in system programming. The development timescales were long, and a high degree of uncertainty pervaded the development process.

About ten years ago, Birman developed the Isis system [2]. Isis is based on message passing within process groups and on causally or totally ordered delivery of messages. It also detects faults and reconfigures the system to remove faulty processes. Isis significantly simplifies the programming of fault-tolerant distributed systems and has had a major impact on the design of such systems, both commercially and in research.

Unfortunately, at least initially, Isis suffered from performance problems that precluded its use in real-time applications. More recent multicast group communication systems, such as Birman's Horus system [5], Dolev's Transis system [3], and our own Totem system [4], are very efficient, and can provide reliable totally-ordered multicast messages with no greater cost

than is required for point-to-point messages. Low cost totally-ordered messages make the design and implementation of fault-tolerant distributed systems significantly simpler than in Isis where efficiency required the use of causally ordered messages.

At the University of California, Santa Barbara, during the development of our Totem multicast group communication system, we have built several demonstration fault-tolerant systems, such as a highly-simplified air traffic control system, demonstrated at the 25th Annual Fault-Tolerant Computing Symposium. These systems were built by M.S. students, typically quite capable and probably better than the average industrial-strength application programmer.

It is our judgment, having watched these students building their demonstration systems, that the message-passing group communication paradigm is still too complex and difficult, and will preclude the widespread development and application of fault-tolerant distributed systems. Too much of the maintenance of consistency of replicated information, too much of the fault detection and recovery, too much of the membership and induction of new members, is visible to the application programmers. These are very difficult problems, difficult even for expert programmers and experienced researchers; they should not be programmed by application programmers.

We are now entering a new era in computing, particularly in real-time computing. In the past our applications have been limited by inadequate memory, by inadequate processing speed, and by inadequate communication bandwidth. Modern computer and communication hardware technology have already largely eliminated these limitations, and improvements in the performance of hardware continue apace, as do reductions in its cost. Meanwhile, the complexity of our application software has increased substantially, greatly increasing the difficulty of its development. It is un-

This work was supported by DARPA Contract N00174-95-K-0083.

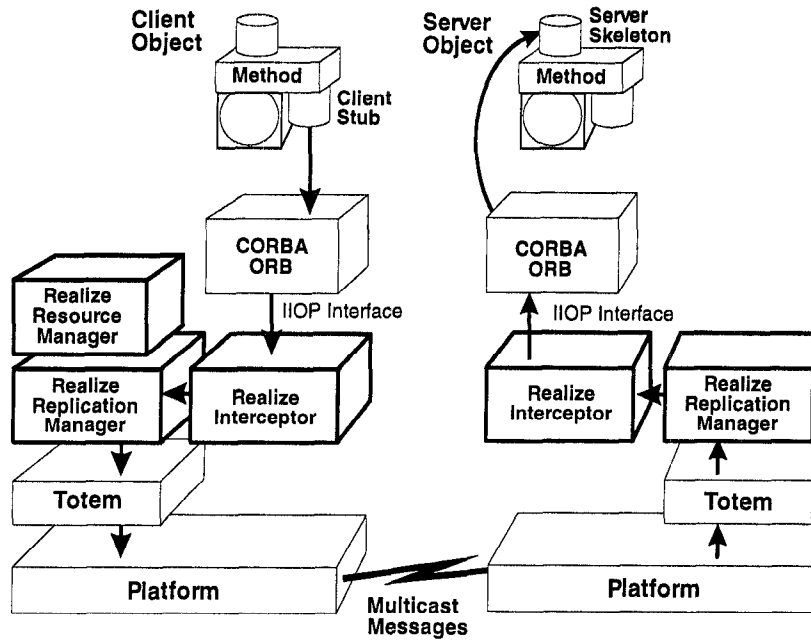


Figure 1: Structure of the Realize System.

doubtedly true that some of our real-time applications will be able to use every processor cycle and every communication bit that we can make available, but it is clear that the primary limitation on future fault-tolerant real-time systems will be the development of the application software, rather than the availability of hardware resources.

It is our belief that, to make distributed systems and fault tolerance widely available, we must be able to separate the programming of the functionality of the application from the programming of the efficient management of resources. The functional behavior of the application must be programmed by programmers who understand the application. The programming of fault tolerance, of maintaining the consistency of replicated information, of assigning objects across a distributed system, of load balancing and real-time scheduling should be undertaken by programmers with expertise in those specific areas.

2 The Realize System

Consequently, at the University of California, Santa Barbara, we are moving beyond message-based multicast group communication protocols towards a paradigm in which the application programmer writes a CORBA object-oriented program as though it were to run on a single processor. All of the object replication,

distribution, consistency and fault tolerance, and indeed all of the message passing, are hidden from the application programmer. Undoubtedly, we will incur additional processing costs compared to a custom-crafted fault-tolerant distributed system, but processors are cheap and becoming cheaper, and programmers are expensive and becoming more expensive. For many systems the most important criterion is the timescale to fielding a reliable, effective application system.

The structure of our system, called Realize, is shown in Figure 1. At the top, an object-oriented application program runs above a CORBA ORB. Real time and fault tolerance are still quite small markets, and they also present substantial difficulties. Consequently, it is unlikely that a major commercial ORB vendor will seek to address them. Conversely, it is unlikely that a customized ORB, developed specifically for fault tolerance and real time, will be able to match a commercial ORB in efficiency, features, and ease of use. It is therefore important that our approach should be able to use existing commercial CORBA ORBs straight out of the box without any modification.

Fortunately, all ORBs are designed to provide the IIOIP interface that permits interworking among ORBs from different vendors or operating on different platforms. The IIOIP interface is simple and tightly defined to ensure interworking. The ORB invokes TCP/IP to communicate the messages of IIOIP, but the Realize

Interceptor mechanism intercepts those calls to TCP and redirects them to the Realize Replication Manager. The Interceptor mechanism is derived from the Catcher program of Schauer [1] at UCSB, and uses the Unix /proc command which was originally intended for debugging and monitoring.

The Realize Replication Manager multicasts operations to multiple replicas of objects, using the Totem group communication system to achieve reliable totally ordered multicasting of messages at a cost no higher than that of point-to-point TCP/IP message transmission. Communication is still possible with unreplicated objects, supported by ORBs that have not been extended by Realize. Communication with such objects continues to use IIOP over TCP/IP.

The Replication Manager is responsible for ensuring that all replicas of an object remain consistent, for which it uses the total order on messages provided by Totem. The Replication Manager is also responsible for fault recovery following a fault. Either active or passive replication can be used as appropriate for each object. The replication of objects is completely invisible to the application, and indeed even to the ORB. They do not need to know how many replicas exist, which type of replication is being used, or where the replicas are located. The object-oriented program is written as though it were to run on a single computer without any replication.

To the left of Figure 1 is the Realize Resource Manager. The objects of the application program are written, and execute, without knowledge of how they are distributed across multiple processors. The IDL specifications can be extended to define the resource requirements of each object and the quality of service requirements of the application. Quality of service includes real-time response, availability, reliability, and security. It is the responsibility of the Resource Manager to assign replicas of objects to processors so as to ensure that the quality of service requirements are satisfied. The Resource Manager knows nothing about the functionality of the application program; it is aware only of the application program's resource requirements. Similarly, the application program is not complicated by consideration of resources and quality of service; its sole concern is functionality.

As yet, we do not have adequate languages available for describing the quality of service requirements of the application, and descriptions of the resource requirements of applications are still quite primitive. Ultimately, we hope, such requirements will be expressed in an extension of IDL and will be stored in the CORBA Interface Repository, from which they will be retrieved by the Resource Manager.

3 Conclusion

A popular topic of research is the investigation of mechanisms by which an application program can be informed of the current availability of resources and quality of service, and can adapt either its own behavior or that of the infrastructure to optimize its performance in the current context. This research is motivated, of course, by the inadequate communication infrastructure currently available and by the desire to improve the performance of applications operating over that inadequate infrastructure.

We believe that this approach, however necessary it may be in the short term, is inappropriate in the long term. It causes the application programs, which are already too complex, to become yet more complex. To us the only acceptable quality of service is a service that the application programmer can just assume and about which he or she does not need to worry. We believe that, in the future, it will be cheaper to provide adequate resources, appropriately engineered and allocated to achieve the quality of service needed by the application. Any increase in program complexity will increase application development cost and timescale. This separation of concerns between the functional application program and its resource management will become increasingly important as applications become more complex.

References

- [1] A. D. Alexandrov, M. Ibel, K. E. Schauer and C. J. Scheiman, "Extending the operating system at the user level: The Ufo global file system," *Proceedings of the 1997 Annual Technical Conference on UNIX and Advanced Computing Systems*, Anaheim, CA (January 1997), 77-90.
- [2] K. P. Birman and R. van Renesse, *Reliable Distributed Computing with the Isis Toolkit*, IEEE Computer Society Press, Los Alamitos, CA, 1994.
- [3] D. Dolev and D. Malki, "The Transis approach to high availability cluster communication," *Communications of the ACM* 39, 4 (April 1996), 64-70.
- [4] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia and C. A. Lingley-Papadopoulos, "Totem: A fault-tolerant multicast group communication system," *Communications of the ACM* 39, 4 (April 1996), 54-63.
- [5] R. van Renesse, K. P. Birman and S. Maffei, "Horus: A flexible group communications system," *Communications of the ACM* 39, 4 (April 1996), 76-83.