Interactive-Group Object-Replication Fault Tolerance for CORBA*

Brent E. Modzelewski, David Cyganski, Ph. D.
Electrical and Computer Engineering Dept., Worcester Polytechnic Institute
Marian V. Underwood
Lockheed Martin Corporation, Government Electronic Systems

Interactive-Group Object-Replication Fault Tolerance for CORBA[*]

Brent E. Modzelewski[†]
Electrical and Computer Engineering Dept., Worcester Polytechnic Institute
David Cyganski, Ph. D.
Electrical and Computer Engineering Dept., Worcester Polytechnic Institute
Marian V. Underwood
Lockheed Martin Corporation, Government Electronic Systems

Abstract

As more and more computers and workstations enter the workplace they are inevitably connected to a network. Networks provide the interconnection necessary for computers to share common data, peripherals and other system resources. Distributed computing allows network applications to access functions or processes on remote computers. Developing network applications specifically to interact and draw upon resources of multiple computers creates the groundwork for a distributed system or distributed computing environment (DCE).

An ideal distributed system is self monitoring and resilient to failures. In the event of a failure the system should dynamically reconfigure itself with automatic fail-over for applications that fall victim to the fault. Transparency of fault tolerant mechanisms is desirable, especially when introducing legacy applications into the distributed system. The reduction of application development efforts heavily relies on the availability of portable, non-invasive, fault tolerance providing extensions, which introduce mechanisms for uninterruptible service by insertion into existing distributed applications.

To test the potential for addressing some of these desired capabilities for a distributed system implemented within a CORBA distributed computing environment, the Interactive-Group Object-Replication (IGOR) system was developed. IGOR is a system of objects that provides fault tolerance through object replication by arranging replicas in fault tolerant groups which interact to provide access to redundant data and services. For purposes of portability, interoperability and to evaluate the CORBA environment, IGOR was designed with the constraint of lying entirely within the CORBA architecture and using IIOP as the communication protocol. This guarantees its portability over changes in platform and network technologies. The IGOR system is reconfigurable and its fault tolerance mechanisms are completely transparent to client applications.

1. Introduction

A plethora of computers and workstations enter the workplace each year and play an increasingly important role as a digital tool used by people worldwide. With the increase of the use of computers comes the increase of stored information and multi-client services. However experience has shown that such information sources and services quickly become decentralized, then isolated and as a result, not interchangeable. Isolation is due to the disintegration of applications segregated by disparate hardware and operating systems, which lack the interoperability and robustness necessary for seamless information and service sharing.

A distributed computing environment (DCE) implies an environment in which interoperation is not only possible, but is fundamentally inherent and natural. Distributed computing exploits the computational power of many computers, integrating the entire system into a single functional unit. Load balancing, parallel processing and distributed objects are major technologies that have evolved which aid in the implementation of full fledged distributed systems.

An ideal DCE provides the programmer with automated tools that permit construction of distributed applications (clients and servers). Distributed applications access functions on remote computers while giving the appearance of locally executed functions.

[*] This research supported by Lockheed Martin Corporation, Government Electronic Systems
[†] Corresponding author: brentm@ece.wpi.edu

Furthermore, the user need not be aware of the remote or local nature of the processes, nor the movement and conversion of the data involved.

Usually the preconceived notion of a distributed system is one in which a specific task is being carried out by a small set of networked computers running the same operating system. That may have been true several years ago, but today a distributed system can be much more diverse. Distributed systems can span many computers and interoperate with virtually any operating system or hardware platform available on today's market. The magnitude of distributed systems varies greatly depending on the intended purpose. A distributed system can contain a few interoperating computers, or may span the entire Internet.

2. Overview

A component or object based architecture can greatly benefit a distributed system. Construction of a system with objects vastly increases modularity; the interchangeable nature of individual system components translates into design and implementation flexibility for the applications engineer. Ideally, upgrades to the system can be done on a component basis, while the system remains up and running. Zero downtime is a very attractive feature of an object based distributed system, especially for mission critical applications. Providing uninterruptible system-wide service is a difficult and complex task. Many standards have been developed to aid in the composition of such systems.

One of the distributed systems standards that has been devised since the emergence of the object paradigm is called the Common Object Request Broker Architecture (CORBA) [2]. CORBA is one component of the Object Management Architecture (OMA) and was developed by the Object Management Group (OMG) [1]. CORBA is a well defined robust standard that is component (object) based, is supported on virtually all hardware platforms, and is fully interoperable and portable.

On the surface, the CORBA standard for distributed objects appears to be another type of Remote Procedure Call (RPC) [3] implementation. On closer inspection, CORBA proves to offer much more than just predefined procedure calls with static parameters. Execution of remote objects, parameter marshaling, multiplatform interoperation, interface portability, dynamic interface invocation, variable parameters and platform independent data types are some of the features of CORBA that do not exist in standard RPC, OSF/DCE or message passing standards, such as Message Passing Interface (MPI) [5] and the like.

While many aspects of CORBA are attractive for large scale distributed system development, it does not inherently support more than rudimentary levels of fault tolerance. To implement basic fault tolerance in CORBA without involving external mechanisms, server applications can be cloned and distributed throughout the network to provide high availability of services to clients. Cloning merely creates redundant copies of the server application, so services are more readily available to client applications. Unfortunately this does not take into account the mutable state of the application at hand. Since no data synchronization takes place between clones, this presents a low level of fault tolerance and is unacceptable for many fault critical applications.

During failures, certain CORBA Object Request Brokers (ORBs), such as Visigenic Software's VisiBroker [6], can automatically fail-over to an application that can provide a desired service. However there is no guarantee of appropriate object state. Fail-over occurs transparently to client applications, thus providing a layer of isolation between the client and the system's fault tolerant mechanisms.

Replicating server applications is another technique similar to cloning that provides high availability of services to clients. Object replication is meant to not only provide availability of services, but also to maintain strict data consistency between objects [4].

3. IGOR Architecture

In response to the need for a fault tolerant system and our desire to test CORBA with respect to its support for easily insertable object behavior extensions, we have developed the IGOR (Interactive-Group Object-Replication) system. Object replication was selected as the basis for implementation of IGOR's fault tolerance mechanism. IGOR is a system of interacting objects that provides mechanisms for the creation of a reconfigurable fault tolerant system, with the additional and strategically important constraint of lying entirely within the CORBA architecture. Layered on CORBA, IGOR yields a portable, interoperable and modular design, that will remain portable over changes and improvements in the CORBA standard and changes of platform, operating system and communication technologies.

Much of our attention has been directed towards development of a fault tolerant system which reduces the invasiveness of the underlying fault tolerant mechanisms with respect to implementation and operation. Our aspiration for IGOR was to provide a tool that would ease the development of fault tolerant distributed applications, while simultaneously embracing the introduction of legacy (CORBA and non-CORBA) applications into the fault tolerant arena.

An object grouping scheme has been devised for the IGOR system to facilitate fault tolerance by redundancy (object replication). Distributed replica server applications enroll themselves into fault tolerant groups through an IGOR registration process. These groups are actually logical representations that associate like server applications that share a common data set. Each fault tolerant group functions separately as a single logical unit and group members interact with each other to maintain intragroup data consistency. Client applications benefit by the group's high availability of services and redundant data.

Fault tolerant groups in IGOR are resilient to partial failures and provide these fault tolerant services transparently to client applications. In fact, the client object does not need to be aware that the server application which it is accessing is a member of a fault tolerant group. The client code is identical whether the client object is connected to an IGOR fault tolerant object or a single non-fault tolerant object. Because of this flexibility, fault tolerance may be added to the system even after client applications have already been deployed. This is done by simply replacing the non-fault tolerant server objects with their IGOR fault tolerant counterparts. No code modification or recompilation of the client application is required for the addition of IGOR fault tolerance.

A single IGOR Registry Service acts as the governing body for fault tolerant group enrollment. The purpose of the registration process with the IGOR Registry is to ensure that fault tolerant groups consist of only objects of identical type. The IGOR Registry is responsible for recording the logical arrangement and association of all replica groups; of course this information is persistently stored concurrently in a redundant object database. To keep track of all the fault tolerant replica groups the Registry constructs a binary tree consisting of all groups, this binary tree being called the Group Tree. A single Group Tree represents active fault tolerant groups for the entire IGOR system. The purpose of the Group Tree is only to facilitate organizing replica groups in a meaningful fashion to ease the Registry's task of group management. To arrange group members, each node on the Group Tree contains a balanced binary sub-tree of replicas for the group; this sub-tree is called the Object Tree. An Object Tree logically represents the group membership of a single fault tolerant group.

In the event of a Registry failure, a new Registry is launched and retrieves fault tolerant group information from the object database. To further protect the system, each member of a fault tolerant group caches a local copy of information regarding the current status of the group memberships. This decouples the fault tolerant groups from the Registry, therefore, the system can continue to operate even in the absence of the Registry.

A set of IGOR objects harboring methods for fault tolerance are integrated into server applications, this alleviates the burden which would otherwise be placed on the server application to implement all fault tolerance mechanisms. These IGOR objects handle fault tolerant group membership related functionality, object monitoring and perform intragroup communication transparently from within the server application. IGOR objects have their own CORBA interfaces and converse amongst each other to perform maintenance tasks, such as reconfiguration and message propagation. Intragroup propagation uses the branches of the Object Tree (binary tree) as communication paths to ensure a single and complete group propagation of all messages. The Object Tree evenly distributes the burden of messaging to all group members. The responsibilities of transaction processing is shared between the IGOR objects and the server application. The Two-Phase Commit [3] protocol was used for the transaction processing associated with object state transfer among replicas.

A system should not render itself inoperable because of a partial network failures or a few downed computers. Fortunately, IGOR's redundant component design protects against such problems by automatically reconfiguring itself when failures occur. This is possible since the IGOR system is self monitoring and can quickly detect problematic objects and adjust accordingly.

4. Implementation Experience

Originally, the intention was to have the server application inherit fault tolerance mechanisms through a standard IGOR Object (C++ base class) with a CORBA interface. The inheritance approach proved to be infeasible due to some limiting constraints of the CORBA standard. Inheriting the IGOR Object class would entail inheriting both its IDL interface and the code associated with its fault tolerant mechanisms into a server application (which has its own IDL interface and associated code). In its current state, CORBA does not support multiple implementation interface inheritance within a single object. CORBA does support multiple interface inheritance within IDL, however, applications cannot inherit from multiple implementations of interfaces.

Consequently, we decided to incorporate the IGOR Object within the server application as a class member. Although not the original intent, inclusion of the IGOR Object as a class member yields a tightly coupled link between the server application and the IGOR Object. As a result, management of fault tolerant operations are largely performed by the IGOR Object on behalf of the server application, despite the inability to directly inherit such functionality. Coupling of the IGOR and server objects fuse the two objects together with interaction between them mediated by standard C++ method invocations. On the other hand, each object uses its own CORBA interface for remote communications.

As much fault tolerant code as possible has been offloaded to the IGOR Object. Unfortunately all the fault tolerant code cannot be handled by the IGOR Object alone. Certain aspects of server specific information (server's mutable state) must be handled by the server application in conjunction with the IGOR Object's involvement. The server's object state type cannot be known to the IGOR Object. Although the IGOR Object plays an important role in maintaining the data synchrony of the fault tolerant group, it does it somewhat blindly with respect to the actual data that is being transmitted. The IGOR Object knows nothing of the server's data, but it takes control of moving the data by instructing the server with respect to where to send or get state information. Again, intragroup state exchanges follow the branches of the Object Tree for communication.

Installing IGOR fault tolerance into a server application requires inheritance of a transaction processing class and creation of additional proxy methods to aid the IGOR Object in intragroup data transfer and transaction processing. Such proxy methods would not be required if direct object implementation inheritance was possible. Also, an object state class called StateKeeper is inherited, which contains methods for mutex locking and unlocking to protect the object's state from multiple thread access.

5. Conclusion

Conception of IGOR was a result of the need for fault tolerance in a standard operating environment. CORBA provides the means to realize fault tolerant distributed systems; IGOR capitalizes on the power and flexibility of CORBA to provide tools to aid in the creation of a fault tolerant system.

We were successful in completely isolating the client applications from all fault tolerance mechanisms by embedding fault tolerance functionality into the server applications, using pre-defined IGOR objects to perform a majority of the work.

Hopefully, CORBA's evolution will encompass the capabilities that make the design of fault tolerant systems less complex and less performance costly.

References

[1] Robert Orfali, Dan Harkey and Jeri Edwards *The Essential Distributed Objects Survival Guide*, John Wiley & Sons, Inc. 1996.

[2] Object Management Group *The Common Object Request Broker Architecture and Specification*, Revision 2.0, July 1995.

[3] Jean Bacon *Concurrent Systems: An Integrated Approach to Operating Systems, Database, and Distributed Systems*, Addison-Wesley Publishing Company, 1992.

[4] George Coulouris, Jean Dollimore and Tim Kindberg *Distributed Systems, Concepts and Designs*, Second Edition. Addison-Wesley Publishing Company, 1994.

[5] University of Tennessee *MPI: A Message-Passing Interface Standard*, May 5, 1994.

[6] Visigenic Software, Inc. *VisiBroker for C++ Programmer's Guide*, October 1996.