Experiences Using DCE and CORBA to Build Tools for Creating Highly-Available Distributed Systems

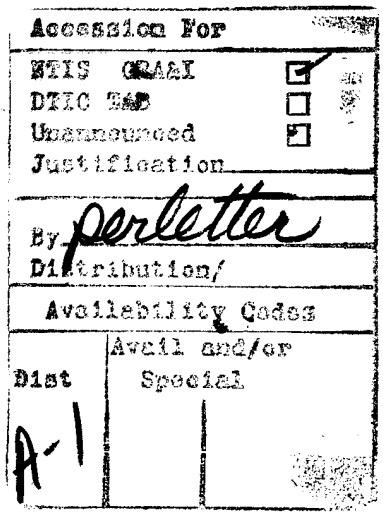
E.N. Elnozahy[1] V. Ratan[2] M.E. Segal[3]

February 1995
CMU-CS-95-117

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

To appear in the International Conference on Open Distributed Processing, February 1995.

[1]Dept. of Computer Science, Carnegie Mellon University, Pittsburgh , PA 15213. Author was supported by a National Science Foundation Research Initiation Award under grant number CCR-9410116.
[2]Dept. of CSE, FR-35, University of Washington, Seattle WA 98195.
[3]Bellcore, Morristown, NJ 07960
The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the NSF, the U.S. Government, Carnegie Mellon University, University of Washington, or Bellcore.



Keywords: CORBA, Distributed Systems, DCE, Fault tolerance, High-Availability, Object-Oriented Systems, Standards

Abstract

Open Distributed Processing (ODP) systems simplify the task of building portable distributed applications that can interoperate even when running on heterogeneous platforms. In this paper, we report on our experience in augmenting an ODP system with tools that allow developers to build highly available distributed objects with little or no additional programming effort. Our tools are implemented within the context of the DCE and CORBA standards for distributed computing. We describe the system that we built and how the combination of DCE and CORBA often helped our efforts and sometimes impeded them. Based on our laboratory experiences, we conclude that these standards generally have a good potential for developing tools for high availability that are portable and applicable to a variety of applications in a distributed computing environment. This potential, however, is hampered by several shortcomings and problems in the specifications of the standards. Such problems could impede other developers and researchers who plan to use these standards. We discuss these problems and suggest solutions to them.

1. INTRODUCTION

Open Distributed Processing (ODP) systems reduce the complexity of designing and implementing applications in distributed computing environments. Applications that run on an ODP system follow standards that allow them to be portable across heterogeneous platforms, and also allow them to interoperate with other distributed applications that follow the same standards. This paper describes our experiences augmenting an ODP system with a toolset that can automatically add high availability to distributed objects. These objects adhere to the Common Object Request Broker Architecture standard (CORBA) [OMG91, Vinoski93]. A highly available object continues to run in the presence of hardware or software faults, as well as planned maintenance activities such as hardware and software upgrades. Applications where high availability is important include financial transaction-processing systems, telecommunications, medical systems, and real-time process control.

The toolset includes a number of software-based techniques for providing high availability with little or no intervention from the programmer. Application developers implement their objects following the CORBA standard and link them with our toolset to provide the required high availability. The implementation uses a locally-developed CORBA library [Diener94] which runs over OSF's Distributed Computing Environment (DCE) [Millikin94, OSF91] on SparcStations running SunOS 4.1 and DEC Alphas running OSF/1.

We chose a CORBA-compliant platform to implement our toolset because we believe that many future distributed applications will adopt the CORBA standard. Thus, our toolset could be ported to other platforms and application domains. We decided also to rely on DCE to provide the networking support. There are several alternatives to this decision, each typically consisting of a CORBA package that implements its own name service, and interacts with the network directly through the socket layer in Unix® or using another RPC system. We decided against using these CORBA packages because their reliance on non-standard naming, and lack of security facilities. DCE does not have these problems and complements the CORBA library that we had with its naming, security, and RPC services. Our choice also offers a potential for interoperability with other "pure" (i.e. non-CORBA) DCE applications and tools.

The implementation of our toolset was able to benefit from many of the facilities that CORBA and DCE provide, such as the name server, the uniform Interface Definition Language, and RPC groups, among many others. These contributed to the simplification of the implementation effort and we were able to verify the benefits that both standards offer for the development of distributed applications. Unfortunately, our laboratory experiences revealed a number of problems with both CORBA and DCE. Though some problems were specific to our platform, others were resulting from the definitions of both standards and could impede other researchers and developers who would be involved in projects using CORBA and/or DCE. We discuss these problems and we suggest solutions to them.

The primary focus of this paper is our experiences using CORBA and DCE to build a high-availability toolset. A detailed description of the implementation of the toolset itself and a performance evaluation can be found elsewhere [Elnozahy95]. Section 2 includes an overview of the high availability toolset to provide the necessary background and context for describing our experience with CORBA and DCE, which is detailed in Section **3**. We present a summary and our conclusions in Section 4.

® Unix is a registered trademark of Novell, Inc.

2. The Design of a High Availability Toolset

2.1. Design Goals and Overview

We have constructed a toolset that provides high availability to distributed applications with little or no additional support by the application programmer. This approach relieves the programmer from the mundane and often error-prone tasks of handling failures and recoveries at the application layer. Also, the approach has the potential of improving the availability of existing applications that were written without consideration for high availability.

The system uses a local implementation of the CORBA standard, called Touring Distributed Objects (TDO) [Diener94]. In this system, application programs consist of distributed server and client objects that communicate by remote method invocation according to the CORBA standard [OMG91]. Server objects follow a multithreaded programming model, where a thread is automatically started to execute the method invoked by a remote client object. It is assumed that the execution of a method invocation is short and the corresponding thread lives only during the course of serving the invoked method. This model is consistent with the familiar remote procedure call paradigm for client/server applications.

TDO objects are written in C++ and use CORBA's Interface Definition Language (IDL) to define the exported methods. Figure 1 illustrates how the components that make up a TDO CORBA/DCE application fit together.

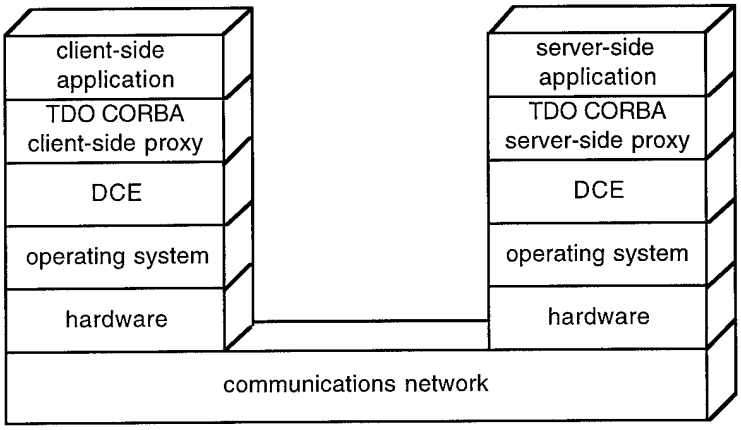


Figure 1 — The Structure of a TDO CORBA/DCE Application

Figures 2a and 2b show the C++ source code for a simple TDO CORBA/DCE application. The TDO runtime system provides support for exporting the server methods, implemented by server implementation (SI) objects, by assigning a unique name in a hierarchical name space for each instance of an exported class. The TDO compiler automatically generates two proxy classes, one for the client side and the other for the server. These classes act as stubs of remote communication for each IDL interface. The server proxy class (SP) handles all the details of translating the remote procedure calls into C++ method invocations, and of exporting the name of the class into the global name space. Client objects that wish to interact with a certain server must be linked with the corresponding client proxy of the server. The client can then access the server through normal invocations of C++ methods. The client proxy locates the required server and handles the details of translating the remote invocations into remote procedure calls.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.