Distributed Objects

Wolfgang Emmerich
Dept. of Computer Science
University College London
London WC1E 6BT,UK
w.emmerich@cs.ucl.ac.uk

Neil Roodyn
Cognitech Ltd
City Cloisters, 188-194 Old Street
London EC1V 9FR, UK
neil@cognitech.co.uk

ABSTRACT
This tutorial motivates the need for, and discusses the principles of object-oriented distribution middleware. We will give an overview how these principles are supported by the major incarnations of object-oriented middleware, the Object Management Group's Common Object Request Broker Architecture (CORBA), Microsoft's Distributed Component Object Model (DCOM) and Java's Remote Method Invocation (RMI). We will discuss common design problems that occur when building applications using distributed objects and present techniques for their solution.

1 INTRODUCTION

In a number of domains, such as the financial and telecommunications sector, applications have to be adapted to evolving market conditions so quickly that it is no longer feasible to develop and maintain large, centralised applications. Applications are now "down-sized" into manageable components, which can be developed and maintained autonomously and be distributed over an enterprise network.

Components are rarely used in isolation. They rather have to be integrated into enterprise application infrastructures. This often involves the integration of legacy applications that cannot be changed and ported to new platforms. Again, this demands integration of new components with legacy systems that are distributed over the network of the enterprise.

Building a distributed system is more complicated than building a centralized system; distribution imposes several challenges to any application development. Autonomously implemented application components tend to be heterogeneous as they are usually implemented in different programming languages and are targeted for different hardware and operating system platforms. If application components are distributed over a network, problems such as component location and re-location, asynchronous communication between components, security and concurrency control, must be addressed. In addition, any distributed application must be tolerant of failures, such as temporarily unreachable components or network time-outs. Dealing with these issues at the network operating system layer is overly complicated for an application engineer.

Distribution middleware is layered between network operating systems and distributed applications in order to bridge this gap. Middleware provides the application designer with a higher-level of abstraction. Middleware systems implement this higher level of abstraction based on primitives that are provided by network operating systems. While doing so they hide the complexity of using a network operating system from application designers. The idea of middleware is not new. Many different types of middleware have been suggested, been built and are being used in industrial practice. This tutorial provides an overview of object-oriented middleware approaches. They all have in common that they implement the communication between distributed and possibly heterogeneous objects.

2 TUTORIAL CONTENTS

Principles

All object-oriented middlewares implement *object requests*, that is they enable a client object to request an operation execution from a server object. Object-oriented middlewares generally have an *interface definition language* that is used to define the interfaces of server objects. Interface definitions may be programming language independent. In this case, they have *programming language bindings* that are defined to map programming languages used for client or server object implementation to the interface.

Object requests are generally executed synchronously; the client object is made to wait for the operation to execute. Client objects can be located on different machines as server objects. To implement remote requests, middlewares therefore have to utilize network protocols. We discuss how middleware implement the session and presentation layers of the ISO/OSI reference model. We review *object identification* and *object activation* that are part of the session layer implementation. We then discuss presentation layer implementation techniques for *marshalling* and *unmarshalling*. These are used to transmit complex request parameters across the network and to resolve data heterogeneity.

OO Middleware Systems
The tutorial uses three examples to show how the above principles are applied in practice. We discuss the Object Management Group's *Common Object Request Broker Architecture* (CORBA) [4, 5], which was the first object-oriented middleware. While CORBA is an open specification that is implemented by a different object request broker, Microsoft pursues a proprietary middleware approach. Microsoft's Component Object Model (COM) [1] supports distributed communication using a specification known as *Distributed COM* (DCOM) [2]. We then present *Remote Method Invocation* (RMI) in Java [3]. Java/RMI is the latest middleware that implement requests between Java objects that reside in different virtual machines. We finally compare CORBA, DCOM and Java/RMI.

Genericity
Object requests can be defined *statically* at the time when a client object is compiled, or *dynamically* when a client object is executed. We compare these two approaches and delineate how they are implemented using CORBA, DCOM and Java/RMI. We suggest when to use static or dynamic requests. To be able to define requests dynamically requires access to *run-time type information*. We review the CORBA interface repository, DCOM's Type Library and Java's reflexivity API and demonstrate how type information that is needed for dynamic requests can be obtained.

Non-Synchronous Communication
By default, one client object requests the synchronous execution of one operation from one server object. Object-oriented middlewares, however, support giving up any of these assumptions. Firstly, they provide non-synchronous forms of communication. We discuss *oneway requests* and *deferred synchronous executions* in CORBA. We show how asynchronous requests can be executed in any object-oriented middleware using multi-threading. Secondly, some middleware supports *request multiplicity*. We show how dynamic requests in CORBA can be used to request execution of multiple operations at once. Finally, we discuss the *group communication* primitives that in CORBA, DCOM and Java. These primitives enable requests between more than two objects.

Distributed Object Life Cycle
The life cycle of distributed objects is considerably more complicated than the life cycle of local objects. Object creation has to determine in addition to how the object is initialized also where the object is to be created. This is supported by all middleware systems using the concept of *factories* and *factory finders*. Server objects might have to be migrated from one machine to another and we discuss how this object *mobility* is achieved in CORBA, DCOM and Java. Finally, deleting objects is more complicated in a distributed setting than in a local run-time environment. Finally, we compare the different approaches to object deletion of CORBA, DCOm and Java.

Persistence
Distributed objects that have an internal state often have to save that state on persistent storage before they are deactivated. We discuss different techniques for implementing persistence. We discuss serialization and externalization that are available in Java. We review the structured storage interfaces for COM and analyze persistence that may be achieved through integration with object databases.

Object Transactions
Sometimes it is necessary to cluster multiple object requests into units called transactions whose execution is atomic, consistency preserving, isolated and durable. We discuss the two-phase commit protocol that is the basic mechanism to implement transactions that span across multiple distributed objects. We present the X/Open Distributed Transaction Processing standard that standardizes an API for two-phase commit and that is implemented by most database management systems. We then review how the CORBA Object transaction service and Microsoft's Transaction Server product implement these two-phase transactions.

Security
The fact that publict networks may be used for the communication between distributed objects make them volatile for security attacks. We discuss different forms of security attacks, such as eavesdropping, message tampering and masquarading. We discuss security mechanisms that are available to avoid these attacks including authentication, access control, auditing, non-repudiation. We review how these primitives are provided in CORBA, DCOM and Java.

3 AUDIENCE
The tutorial is designed for an audience with intermediate experience level. It is intended for software engineering practitioners and researchers, who have basic knowledge of object-oriented design and programming languages, such as C++ and Java, and want to learn how object-orientation can be used to build distributed and heterogeneous software applications.

REFERENCES

[1] D. Box. *Essential COM*. Addison Wesley Longman, 1998.

[2] R. Grimes. *DCOM Programming*. Wrox, 1997.

[3] Javasoft. *Java Remote Method Invocation Specification*, revision 1.50, jdk 1.2 edition, Oct. 1998.

[4] *The Common Object Request Broker: Architecture and Specification Revision 2.0*. 492 Old Connecticut Path, Framingham, MA 01701, USA, July 1995.

[5] *CORBAservices: Common Object Services Specification, Revised Edition*. 492 Old Connecticut Path, Framingham, MA 01701, USA, March 1996.