;login:

THE MAGAZINE OF USENIX & SAGE

February 2001 • volume 26 • number 1

inside:

PRITHVI RAO:
USING CORBA WITH JAVA:
A MINI NAPSTER, PART 1

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

using CORBA with java

A Mini Napster

Part I

by Prithvi Rao

Prithvi Rao is the co-founder of KiwiLabs, which specializes in software engineering methodology and Java/CORBA training. He has also worked on the development of the MACH OS and a real-time version of MACH. He is an adjunct faculty at Carnegie Mellon and teaches in the Heinz School of Public Policy and Management.

<prithvi+@ux4.sp.cs.cmu.edu>

Introduction

The need to understand middleware technologies such as CORBA, DCOM, and RMI has been hastened in recent years partially because they have all matured to the point that they are all capable of being deployed in scalable and evolvable distributed applications.

In this article I present a code example that I call a "mini napster." In this example the client and server communicate using the Object Request Broker (ORB) that is available with the JDK1.2 release. The example itself is a simple Java program, but it is adequate for the purposes of demonstrating the capabilities of CORBA.

A Brief History of CORBA and DCOM

The CORBA movement was largely a response to the pioneering effort by Microsoft in the development of their component object model (COM). In both cases these software capabilities (also known as middleware) made it possible to write powerful distributed applications with more ease. I am not suggesting that writing distributed applications using DCOM and CORBA are trivial but that they are much easier than programming at the remote procedure call (RPC) layer.

In fact one of the stated goals of the Object Management Group (OMG) that developed the specification for CORBA was to make programming-distributed applications as simple as writing non-distributed applications. The salient steps are:

1. Create an object
2. Make it distributable
3. Make it distributed

This approach is predicated heavily on the deployment of sound object-oriented software engineering design and analysis. In our example we will assume that this is the case.

The main difference between DCOM and CORBA is that CORBA has been proven to run on various flavors of UNIX as well as Windows; DCOM clearly runs best on Microsoft platforms, and although the marketing literature suggests that it is supported on UNIX (Bristol and MainSoft are examples of companies making these claims), it is likely that the performance will be unacceptable for most practical purposes.

The CORBA Interface Definition Language (IDL)

The CORBA IDL is a purely declarative language designed for specifying programming-language-independent operational interfaces for distributed applications. The OMG specifies a mapping from IDL to several different programming languages including C, C++, Java, ADA, COBOL and SmallTalk. For each statement in the IDL there is a mapping to a corresponding statement in the programming language. For instance, all the primitive types in Java are supported. There is also provision to define new types such as structures.

One of the main features of the CORBA IDL is that it is intended to capture the design of the server. In other words, the IDL is a language-independent representation of the server and therefore promotes an important concept of "design portability"; conse-

quently it is possible to write the client in one language and the server in another (by using IDL compilers for both languages) and thus promote inter-operability as well.

Napster Server

The Napster server permits a client application to register the name of an artist and album and perform operations on this data. Specifically the operations are:

- Add an item
- Delete an item
- Find an item
- Update an item

Napster also requires that this information be available as a record structure, so it is necessary to define a data type which is a "struct."

The Napster IDL

The Napster IDL file called "Napster.idl" is a text file that has the following entries:

```
module Napster
{
    struct Record
    {
        long version;
        string artist_name;
        string album_name;
        string owner_name;
    };

    interface NapsterServerI
    {
        Record findItemInServer(in string albumName);
        string addRecordInServer(in Record desiredRecord);
        boolean deleteItemInServer(in Record desiredRecord);
        boolean updateRecordInServer(in Record desiredRecord,
        in string newOwner);
    };
};
```

Mapping the IDL to Java

In this section we will compile the Napster.idl file and examine the output. The idltojava compiler takes an IDL file as an input and generates the required Java files as follows:

```
idltojava Napster.idl (or idltojava -fno-cpp Napster.idl)
```

The "module" translates to a Java package name. When this file is compiled using the idl2java compiler it will create a directory called "Napster" into which it adds the client stubs and server skeleton code for use by the client and server.

The "interface" translates to a Java interface that must be "implemented" (recall that you extend classes and implement interfaces). The methods that are defined in this interface are commensurately translated to Java methods in the Java interface.

Making Sense of the Output of IdltoJava

In this section we examine the files that are generated by the idltojava compiler.

NapsterServerIImplBase.java

This abstract class is the server skeleton that provides basic CORBA functionality for the server. It implements the NapsterServerI.java interface. The server class NapsterServant

extends _NapsterImplBase.

NapsterStub.java
This class is the client stub providing CORBA functionality for the client. It implements the NapsterServerI.java interface.

NapsterServerI.java
This interface contains the Java version of the IDL interface. It contains the four methods defined:

```
package Napster;
public interface NapsterServerI
      extends org.omg.CORBA.Object, org.omg.CORBA.portable.IDLEntity {
      Napster.Record findItemInServer(String albumName)
;
      String addRecordInServer(Napster.Record desiredRecord)
;
      boolean deleteItemInServer(Napster.Record desiredRecord)
;
      boolean updateRecordInServer(Napster.Record desiredRecord,
      String newOwner)
;
}
```

NapsterServerIHelper.java
This final class provides auxiliary functionality and, in specific, the "narrow" method required to cast CORBA object references to their proper types.

NapsterServerIHolder.java
This final class holds a public instance member of type NapsterServerI. It provides operations for "out" and "inout" arguments that CORBA has but which do not map easily to Java semantics.

When you write the IDL interface, you are really doing all the programming that is required to generate all the files mentioned to support the distributed application. The only additional work required is the actual implementation of the client and server classes.

In the next article I will present the client and server code for the Napster example and provide instructions on how to run this application. We will observe that the structure of a CORBA server and client code written in Java are identical to most Java applications as described below:

- Import the required library packages
- Declare the server class
- Define a main method
- Handle exceptions

Conclusion

The level of difficulty in writing distributed applications is significantly ameliorated with the advent of middleware such as CORBA. Consider that writing the Napster example using RPC not only requires advanced knowledge of networks and how they work but also promotes embedding network-related code in the application, generally considered bad software engineering practice.

It is true to say that CORBA presents its own challenges and there is a finite learning curve that must be addressed in order to feel confident with this technology. It has been my experience, however, that once the concepts are understood, CORBA will not present any mystery to practitioners.

The level of difficulty in writing distributed applications is significantly ameliorated with the advent of middleware such as CORBA.