Building User Interfaces for CORBA Objects with Java

Michael Ogg, Fabio Previato, and Arun Swaminathan [1]

University of Texas, Austin TX 78712

One of the major advances since the development of the World Wide Web has been the growth in use of the Java programming language. Java allows an application to run on a local machine. The CORBA standard allows a client to invoke methods on remote objects. In this paper we discuss ways in which Java and CORBA can be combined, give examples of their use, and discuss problems, especially in the context of replicated objects. Finally, we show how CORBA and Java are being used in the NILE project.

Key words: Java, CORBA, IIOP, World Wide Web

1 Introduction

One of the most significant developments in Web computing in the last three years has been the introduction and rapid acceptance of the Java programming language [3]. Although most of the features of Java have been used for some time in other languages, Java had the good fortune to offer the right features at the right time. In parallel, the CORBA [10] standard provides a framework for writing object-oriented client-server applications in a language-independent way. That is, as long as a client and server respect the same interface, there are no language constraints for either the client or the server. The growth of Web computing and the importance of client-server computing have made it inevitable that the two would come together as the basis for an open architecture for Web-based commerce. For these same reasons, but perhaps on a more modest scale, the NILE project [2,5] is building its front-end interface with Java to communicate with the CORBA interface of the SiteManager [8]. In NILE however, there is an additional complexity: because of the incorporation of fault-tolerance into the design, any Java GUI has to communicate with a *replicated* object.

[1] The first two authors are supported by the NSF under grant ASC-9318151

2 The Java Language

After the World Wide Web achieved unprecedented growth, there were two options for extending its use from merely displaying information passively. The first was by launching "helper applications" such as `ghostview`, `xdvi`. etc. to display data; this required having the binaries for those applications on the host machine. The second was to use the Common Gateway Interface (CGI) to run applications on the remote host; this had the potential of completely overloading the host. Java overcomes the deficiencies of both these approaches. By using a "Java Virtual Machine", any host that supports Java can run a Java *application*, or any application that supports the Java environment can host a Java *applet*. In this way, a Java program is both machine independent, and runs on the local rather than the remote host.

The main features of Java are well-known, namely: it has simple network-based programming constructs; it has simple and relatively capable graphics constructs; it is extensible with class libraries; and it achieves platform independence by way of interpreted byte code. In addition, its support of threads (as a feature of the language rather than the host operating system) makes it very suitable for asynchronous applications.

3 The CORBA Standard

CORBA implements client-server applications by allowing a client to invoke methods on remote objects. These objects can be in a different process on the same host, or on a different host. In either case, the communication is transparent. A CORBA interface is specified in a language called OMG Interface Definition Language (IDL), which is not so different from C++ class definitions. A particular CORBA implementation has an IDL compiler that translates IDL into a native language (such as a C++ header file), and generates a client *stub* and a server *skeleton*. The actual generated code and header file are specific to the ORB implementation. In the original CORBA specification, there was no requirement for one vendor's ORB to interoperate with another vendor's. With the growth of Web computing, the need for interoperability became obvious, and this was addressed in the CORBA v2.0 specification. CORBA v2.0 still allows native *intra*-ORB communication, but specifies standards for *inter*-ORB communication. The basic standard is the General Inter-ORB Protocol (GIOP); the one of interest here, and the only specific one mandated by CORBA v2.0 is the Internet Inter-ORB Protocol (IIOP). An ORB can also use IIOP for *intra*-ORB communication. The standard specifies how *gateways* and *half-bridges* may be used for translating from an ORB's native communication to IIOP. In particular, if a client is implemented in Java

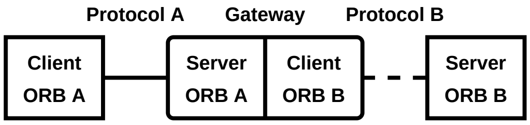

Fig. 1. The Use of a Gateway between two ORB Protocols

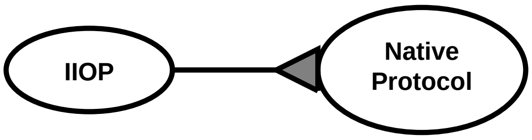

Fig. 2. The Use of a Half-Bridge between IIOP and a Native Protocol

and a server in C++, they can use IIOP for the communication, even if each was developed by a separate vendor. In fact, since CORBA only specifies the object interfaces, there could even be different implementations of the client provided that they all respect the same interface.

4 IIOP with Replicated Objects

One of the features of NILE is its use of replicated objects for fault-tolerance. This is achieved by using an ORB such as Electra [7] which uses a fault-tolerant group communication protocol such as Isis [1] or Horus [11] to ensure that remote method calls are invoked consistently and completely on all replicas of an object. Each object group appears to an external observer, in every way, as if it were a single, fault-tolerant object. If the fault-tolerant paradigm is to be preserved, a single instance of an IIOP gateway is not sufficient, because its crash would leave the service unavailable even if the service itself is still functioning. Technically, such a gateway need not even use IIOP – it simply acts as a server for ORB A and makes client calls for ORB B. This is shown in Figure 1. The requirement is that the gateway must speak both protocols. In contrast, a half-bridge is implemented as a feature of the ORB itself. Messages are received from the outside using IIOP, shown in Figure 2. In the case of Electra, the messages are sent from a single instance of the object group to all members of the group. This feature is expected to be implemented in Electra by the end of February 1997 [6].

5 The NILE Project and a Java GUI Prototype

Two features of NILE make it particularly suitable for using Java and CORBA together: all the external interfaces (which in practice means the interfaces to the SiteManager object) are CORBA interfaces; the client should be light-weight, without any NILE dependencies, and able to work over a wide-area network. This can be realized by using Java clients which have a CORBA interface. Moreover, the other advantages of Java such as its graphics capabilities, and the possibility of using asynchronous call-back, allow the interface to display status information or results dynamically. Because we needed to make progress understanding the capabilities of Java, and the Electra IIOP transport was not yet available, we implemented a prototype entirely in Java using Orbix from Iona Technologies [4], but using IIOP as the transport [9]. For this prototype, the server had all the NILE external CORBA interfaces, but was otherwise a dummy.

In this way, the NILE profile object, described in [2], was constructed and a "job" could be submitted to the SiteManager. There are several advantages to implementing the client as an applet rather than an application. In particular, Java-capable Web browsers such as netscape are ubiquitous and available for most platforms, whereas not so many systems have support for the Java Virtual Machine. Besides, the convenience of using a Web browser is very compelling – the anticipated growth in network computers (or internet appliances) would place almost no restrictions on the client. However, there are implications, mostly concerned with security. For instance: an applet cannot use the client's native file system, whereas a Java application can; an applet can only open a network socket on the remote machine from where it was loaded. In actuality, these restrictions are only policies implemented by Java's security manager and future browsers could have greater flexibility. Nevertheless, both because of today's restrictions and what is probably good policy, we implemented conservative solutions. For storing the NILE profile object, we created a server-side "profile repository". We could extend this concept to include user code or results. This can be thought of as an extension to NILE's Data Repository, and would have the same interface. The socket restriction is solved by enabling an httpd daemon on the server. Since the machines used for the SiteManager will be available and probably not used as SubJobProcessors, this is not really restrictive.

Even though the SiteManager was simulated, we were able to implement all the salient features of the client: job submission, SiteManager query, retrieval of results, notification of events, and continuous update of status. Since the client was written using the IIOP protocol (although this is essentially transparent to the actual Java code), we do not anticipate significant difficulties when Electra supports IIOP and we are able to use the real SiteManager object.

As well as the graphical interface, it is also useful to provide a text-only interface. This might be used if someone only has access to a terminal emulator. This will also be written in Java using the same techniques. In fact, it almost comes "for free" when writing the GUI. In this case though, the interface would have to be a Java application requiring that the target host support the Java Virtual Machine.

6 Conclusions

The acceptance of the Java programming language, and the availability of CORBA v2.0 capable ORBs is opening a new era in Web computing. The interest in the use of the internet for commerce is driving these developments. However, the same advantages are also applicable to distributed applications in High Energy Physics, which is after all where the World Wide Web originated.

References

[1] K. P. Birman and R. van Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, 1994.

[2] D. G. Cassel and M. Ogg. Nile: Distributed Computing and Databases for High Energy Physics. In *NSF Grand and Naational Challenges Workshop*, March 1996.

[3] `http://java.sun.com`.

[4] `http://www.iona.com`.

[5] `http://www.nile.utexas.edu`. NSF, ASC-9318151.

[6] S. Maffeis. private communication.

[7] S. Maffeis. *Run-Time Support for Object-Oriented Distributed Programming*. PhD thesis, Universität Zürich, 1995.

[8] F. Previato, M. Ogg, and A. Ricciardi. Progress with Nile. In *Computing in High Energy Physics*, 1997. (to appear).

[9] A. Swaminathan. The Nile Internet Front-end Tool Initiative – An Experiment in Java-driven Interaction with CORBA Systems. Master's thesis, University of Texas at Austin, January 1997.

[10] The Object Management Group. `http://www.omg.org`.

[11] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: A Flexible Group Communication System. *Communications of the ACM*, 39(4):76–83, 1996.