J. of Ultra Comp. and I.T. Vol.1(1), 49-53 (2010). ISSN 2229-3531 (Print)
www.compitjournal.org 2455-9997 (Online)

Distributed remote method invocation

[1]ANIL RAJPUT, [2]NEERAJ BHARGAVA [3]HEMRAJ SINGH THAKUR,
[4]MANMOHAN SINGH and [5]SHIVSHAKTI SHRIVASTAV

[1]Head of Dept. Maths & Computer Science, Sadhuvasvani College,Bhopal (M.P.) (INDIA)
[2]Head of Dept. Computer Application , MDS University Ajmer (Rajasthan) (INDIA)
[3]Lecturer, Department of Computer Science, Kendriya Vidyalaya, Jaipur (Rajasthan) (INDIA)
[4]Lecturer, Department of Computer Science, BIST, Bhopal (M.P.) (INDIA)
[5]Lecturer, Department of Computer Application, SCOPE, Bhopal (M.P.) (INDIA)

(Acceptance Date 24th February, 2010)

Abstract

An emerging trend in the Signal and Image Processing (SIP) community is the appearance of middleware and middleware standards that can be readily exploited for distributed computing applications by the SIP community. High performance computing and High Performance Embedded Computing (HPEC) applications will benefit significantly from highly efficient & portable computational middleware for signal & image processing. Open middleware standards such as VSIPL, MPI. CORBA, Data mining RMI, and Web Services (based on SOAP/XML), offer a unique opportunity for the rapid development of easily maintained HPEC codes that combine portability and flexibility across a number of applications This middleware infrastructure will support the rapid development and deployment of portable, efficient, SIP critical applications that will be of immediate benefit to many. The use of distributed computing technologies for problem solving has been around for many years. The early paradigm of distributed computing has been that of remote procedure calls (RPC). However, in recent years, this paradigm has shifted to the use of remote objects due to the acceptance of object oriented programming practices. Even today web services are built around the concept of messaging and frequently these messages take the form of request/response-type remote procedure calls on remote objects. The existing and emerging standards for performing distributed computing have resulted in several possible middleware choices for the SIP community. This paper focuses on three specific middleware standards for distributed computing, namely: the Common Object Request Broker Architecture (CORBA).

Index Terms: Remote method invocation, Object request broker, Inter-object communication, Distributed embedded computer systems, Distributed object models, data mining AI.

Distributed computing with RMI-

Remote Method Invocation (RMI) technology, first introduced in JDK elevates network programming to a Higher plane. Although RMI is relatively easy to use, remarkably Powerful technology and exposes the average Data mining developer to an entirely new paradigm—the world of distributed object computing.

RMI Features :

RMI is like a remote procedure call (RPC) mechanism in other languages. One object makes a method call into An object on another machine and gets a result back. Like most RPC systems, RMI requires that the object whose method is being invoked (the server) must already be up and running

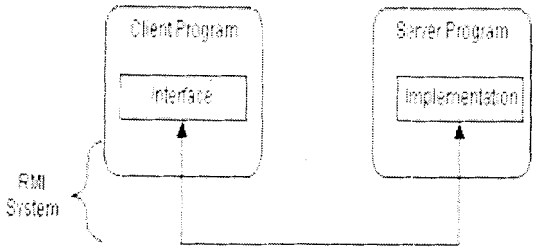
Interfaces : the heart of RMI :

The RMI architecture is based on one important principle: the definition of behavior and the implementation of that behavior are separate concepts. RMI allows the code that defines the behavior and the code that implements the behavior to remain separate and to run on separate JVM's.

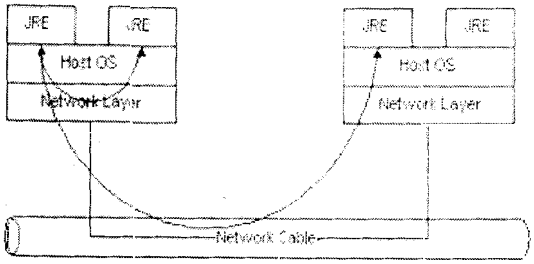
This fits nicely with the needs of a distributed system where clients are concerned about the definition of a service and servers are focused on providing the service.

RMI Architecture Layers :

The RMI implementation is essentially built from three abstraction layers. The first is the Stub and Skeleton layer, which lies just



beneath the view of the developer. This layer intercepts method calls made by the client to the interface reference variable and redirects these calls to a remote RMI service.



The next layer is the Remote Reference Layer. This layer understands how to interpret and manage references made from clients to the remote service objects.

Using RMI·

It is now time to build a working RMI system[5-6] and get hands-on experience. In this section, you will build a simple remote calculator service and use it from a client program. A working RMI system is composed of several parts.

- Interface definitions for the remote services
- Implementations of the remote services
- Stub and Skeleton files
- A server to host the remote services

- An RMI Naming service that allows clients to find the remote services
- A class file provider (an HTTP or FTP server)
- A client program that needs the remote services

Parameters in RMI :

We have seen that RMI supports method calls to remote objects. When these calls involve passing parameters or accepting a return value, how does RMI transfer these between JVMs? What semantics are used? Does RMI support pass-by-value or pass-by-reference? The answer depends on whether the parameters are primitive data types.

Parameters in Single JVM :

First, we will review how parameters are passed in a single JVM. The normal semantics for Data mining technology is pass-by-value. When a parameter is passed to a method, the JVM makes a copy of the value, places the copy on the stack and then executes the method. When the code inside a method uses a parameter, it accesses its stack and uses the copy of the parameter.

Distributing ANG installing RMI software:

RMI adds support[2-4] for a Distributed Class model to the Data mining platform and extends Data mining technology's reach to multiple JVM's. It should not be a surprise that installing an RMI system is more involved than setting up a Data mining runtime on a single computer. In this section, you will learn about the issues related to installing and distributing an RMI based system.

Distributing RMI Classes :

To run an RMI application, the supporting class files must be placed in locations that can be found by the server and the clients. For the server, the following classes must be available to its class loader:

- Remote service interface definitions
- Remote service implementations
- Skeletons for the implementation classes (JDK 1.1 based servers only).
- Stubs for the implementation classes
- All other server classes

For the client, the following classes must be available to its class loader:

Distributed garbage collection :

One of the joys of programming[6,7,8] for the Data mining platform is not worrying about memory allocation. The JVM has an automatic garbage collector that will reclaim the memory from any object that has been discarded by the running program. One of the design objectives for RMI was seamless integration into the Data mining programming language, which includes garbage collection. Designing an efficient single-machine garbage collector is hard; designing a distributed garbage collector is very hard. The RMI system provides a reference counting distributed garbage collection algorithm based on Modula-3's Network Objects. This system works by having the server keep track of which clients have requested access to remote objects running on the server.

Serializing remote objects :

When designing a system using RMI,

there are times when you would like to have the flexibility to control where a remote object runs. Today, when a remote object is brought to life on a particular JVM, it will remain on that JVM. You cannot "send" the remote object to another machine for execution at a new location. RMI make.

CORBA servant sends back a response to a remote ORB client ends a request through its local ORB to a remote orb's servent RMI vs. CORBA :

RMI pros and cons :

Remote method invocation has significant features[1] that CORBA doesn't possess - most notably the ability to send new objects (code and data) across a network, and for foreign virtual machines to seamlessly handle the new objects Remote method invocation has been available since JDK 1.02, and so many developers are familiar with the way this technology works, and organizations may already have systems using RMI. Its chief limitation, however, is that it is limited to Data mining Virtual Machines, and cannot interface with other languages.

Creating 3-Tier Distributed Applications with RMI :

Creating 3-Tier Applications :

The 2-tier model for applications[9,10] is the most common model in use today. Many application designers think only in terms of the database and the application. The availability of 2-tier application builders has helped perpetuate this philosophy. The 2-tier model is not a "bad thing," but there are cases in which the 3-tier model would be a better choice. Just to review, the 2-tier model consists of an application and a database. A 3-tier model consists of an application, a layer of business logic, an. Once you break out of the 2-tier mold, you often start adding multiple tiers. difference between a 2-tier and 3-tier application design. You can also divide your application into an application logic tier and a presentation tier. In a 2-tier model, the business logic is part of the application. In smaller applications, this is not a problem because there may be only one application implementing a particular business process. In larger systems, however, many applications use the same areas of business logic. In a 2-tier environment, this means that the business logic is replicated across every application[8,9,10].

Conclusions

As it is shown in this chapter, RMI is a very complex standard. The RMI architecture is based on one important principle: the definition of behavior and the implementation of that behavior are separate concepts. RMI allows the code that defines the behavior and the code that implements the behavior to remain separate and to run on separate JVM's. Comparing RMI and CORBA doesn't reveal an optimum solution - one is not "better" than the other. The properties of these two technologies lend themselves to different situations. A comparison of RMI and CORBA helps to highlight individual strengths and weaknesses, but the applicability of one technology over the other depends largely on the purposes for which it is to be used, the experience of the developers who will design, implement and maintain the distributed system, and whether non-Data mining systems are intended to

access the system now or in the future.

References

1. Elfwing, R., Paulsson, U. and Lundberg, L., *Performance of SOAP in Web Service Environment Compared to CORBA,* In Proceedings of the Ninth Asia- Pacific Software Engineering Conference, IEEE, (2002).
2. Davis, D. and Parashar, M., *Latency Performance of SOAP Implementations,* In Proceedings of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE (2002).
3. Gorton, I., Liu, A. and Brebner, P., "Rigorous evaluation of COTS middleware technology", *Computer*, IEEE, March, pp. 50-55 (2003).
4. JWSDP Data mining Web Services Developer Pack http:
5. *Tomcatserver:* http://jakarta.apache.org/tomcat/
6. Juric, M.B., Rozman, I. and Hericko, M., *Performance Comparison of CORBA and RMI,* Information and Software Technology 42, pp 915-933 (2000).
7. Juric, M. B., Rozman, I., Stevens, A.P., Hericko, M. and Nash, S., *Java2 Distributed Object Models* In Proceedings of 7th International
8. Zimmermann, O. Tomlinson, M. and Pause, S. Perspectives *on Web Services*, Springer-Vela, Berlin (2003).
9. Orfali, R., Harkey D. and Edwards, J., The Essential Distributed Objects Survival Guide, John Wiley & Sons, Inc. (1996).
10. "Using Rational Rose 4.0", Rational Software Corporation, Santa Clara, Nov. (1996).