RMI based Distributed Computing System

by

Srikanth Koneru (skoneru@vt.edu) Ravikiran Vatrapu (rvatrapu@vt.edu)

CS 5204 Fall 2000

Abstract:

The main purpose of this project is to design and implement a distributed computing system. The objectives of the project are to provide significant price/performance improvement, to inherently incorporate the concept of Jini technology and to provide the choice of selecting a particular service in a set of available services so that the clients (Users) can make well-informed decision(s) on which service to choose.

The source of inspiration for the project was the paper “Jini to the rescue” by Steve Morgan of the ObjectSpace Inc. Created for networking all sorts of electronic devices, Jini lets them join up easily and gracefully. It is plug and play capability for spontaneously forming networks of heterogeneous equipment. The technologies used are Java 2 Platform, which implements RMI and Object Serialization and Jini Technology.

Our learning objective in designing and implementing this system was to gain valuable first-hand insight into the distributed systems in general and RMI, Jini in particular. This project gave us an opportunity to appreciate the theoretical concepts we were taught in the class.
System Model:

The following simplified system model gives a conceptual view of the system.

Client Application Software

Jini Lookup Service

Discovery and Join

Service 1  Service 2  ………………………..Service N
The system interactions can be classified into 3 types.

A. Server service interaction
B. Server client interaction
C. Client service interaction

A. Server-Service Interaction

2. Service sends the’ discovery request’ datagram packets to server.
3. Server permits/doesn’t permit service to join after checking the group validity
4. Server notifies the service
5. Services gets response from server and if accepted it sends the response to the server.
6. Server takes in the response (prog. Interface of new service -- a serialized java object)
7. Server grants a lease to service and service waits for a connection request from the client.
8. Server drops the service when lease expires.

B. Server-Client Interaction

1. Server listens for request from client
2. Client requests for a particular service.
3. Server sends client, list of candidates for that service
4. Client selects one among the list of candidates
5. Lookup service gives the corresponding piece of executable Javabyte code.(which implements the communication/service itself)
C. Client-Service Interaction

1. Service gets a call from server to establish connection with client
2. Client establishes connection with candidate providing the service after server approves.
3. The candidate services the client directly.

Explanation.

When the client wants data and services it goes to a Lookup Service--where it finds directions on how to get to all the services that are willing and able to serve it. The client obtains data from some services and requests processing from others. Once these processes are completed, the client has its desired result. For this we set up an integrated Lookup Service on one host (server1) in the network. Services seeking a market for their services send discovery requests to multicast request servers for Lookup Services. Upon hearing such a request, an interested Lookup Service initiates a private conversation with the service to establish its joining in the market.

When the service and the Lookup Service first converse as part of this joining process, the Lookup Service promises the newcomer space in its "market." In return, the service gives the Lookup Service basic information with which client applications can establish a connection to it.

Client applications ask Lookup Services for certain type of services. The client does this by sending a template file--a generic description of the desired service--to the Lookup Service. The Lookup Service responds with a list of candidates that match the template from which the client may select a particular service. Thus Jini supplies the infrastructure for "marketing" a set of services, as well as provides the programming model by which resources may form impromptu groups (called Jini federations) and provide their services to users and other resources.

Jini is based on Java2 for both its visible interfaces and its on-the-wire protocols. By using Java interfaces, services allow clients to work with known behaviors, independent of their implementations.
Jini's specification depends on the Java serialization mechanism for the transfer of code that is specific to a particular service. As part of its responsibility, the Lookup Service is always listening for the discovery-request datagram packets. Once it hears one, the discovery process begins. If the Lookup Service supports any of the groups requested, the requesting service may be permitted to join. The Lookup Service notifies the requesting service, which responds with the new service's programming interface—a serialized Java object—that the Lookup Service stores.

Since a service may join or leave the network any time, the loose term of "federation" is given to the Lookup Service and its current set of registered services. The newly registered service is granted a lease for its registration, for durations determined by those who initially configure the system. If the lease is not renewed, the Lookup Service will drop the service from the federation.

When a Jini-capable program—that is, a client application—needs a service such as, say, printing, it requests that the Lookup Service send it a list of potential printer services. From the lists provided, the program selects one or more. Jini does not specify how this selection is to be made; it can be done randomly, algorithmically, or even manually through a user interface.

For the selected service provider, the Lookup Service delivers the corresponding piece of executable Java byte code. While most typically this code would simply implement communications between the service and the client program, it could be the actual service itself. [1]

**Package Design.** The simple package level design is shown below.
Class Diagrams

Server1 Package

< Interface> MyServer

- getDataNum(): int
- getData(int n): String
- serviceValidate(int serviceNo): int
- serviceInfo(int serviceNo,String serviceInfo): int
- clientServiceRequest(int serviceType): int
- serviceSelectRequest(int serviceType): int
- clientSelection(int serviceType,int serviceNo): int

Server2 Package

< Interface> MyServer2

- getDataNum() : int
- String getData(int n) :String
- serviceValidate(int serviceNo) : int
- serviceInfo(int serviceNo,String serviceInfo) : int
- clientServiceRequest(int serviceType) : int
- serviceSelectRequest(int serviceType) : int
- clientSelection(int serviceType,int serviceNo) : int
- serverSearchService(int serviceType) : int

Service Package

< Interface> serviceServer

- grantPermission(int clientCode):int
- startInteraction(int clientCode):int
Summary:
This project has been a wonderful learning experience. Understanding the intricacies of RMI was an effort worthwhile. We could fully appreciate the potential and promise of the Jini technology. The project helped us to relate to the class more concretely by identifying areas of concern. The system can be extended to include an accounting system that implements a charging model and security and authentication measures to prevent services that are not genuine. The addition of lease concept on the servers will fully model our system as a marketplace for services and clients interested.

References:
[1] “Jini to the rescue”, Steve Morgan
[2] Java Unleashed, Jamie Jaworski