Event Notifications among Distributed Objects in Dealing Room System

Su Su Aung, Yin Ko Latt
University of Computer Studies, Magway
susuaung87@gmail.com

Abstract

In the online business world, the customer-vendor relationship is no longer mearly buy and sells [7]. As there are various kinds of stock to be chosen on the market, it is important for the customer to be able to choose the best with reasonable price. This paper presents about the implementation of dealing room system which can provide the current market prices of stocks to user and can inform the user immediately when the stock prices change. There are three components in dealing room system. They are information provider, server and user. Information provider is an authorized person who can access the database that stored the information of stock on server. Information provider can insert new stock, delete the existing stock and update the prices of stocks. When the user enters to the system, user can find out the current prices of stocks and the date of the latest update. Dealing room can also let users to register their interested stocks with the server in order to get the notification when the price of user interested stock is changed or deleted. In this system, server stores the stock information and send notification message to user when information provider updates the database.

1. Introduction

Most of the programming languages today are object-oriented. With the widespread utilization of object technologies, it has become more and more important to employ the object-oriented paradigm in distributed environment [6]. The distributed object system is simple, in that distributed objects are easy to use and to implement. And the system it self is easily to extensible and maintainable. Thus distributed object-oriented platforms have become important components for parallel and distributed computing and service frameworks. Among distributed object-oriented software, RMI is one of the key methods for performing parallel and distributed computing in Java environments. The java system provides a basic communication mechanism called sockets. Although sockets are flexible and sufficient for general communication, the use of sockets requires the client and server using this medium to engage in some application-level protocol to encode and decode messages for exchange. Design of such protocols is difficult and can be error-prone. An alternative to sockets is Remote Procedure Call (RPC). RPC systems abstract the communication interface to the level of a procedure call. RPC, however, does not translate well into distributed object systems where communication between program-level objects residing in different address spaces is needed. In order to match the semantics of object invocation, distributed object systems require remote method invocation or RMI. In RMI there are remote objects whose methods can be accessed from another address space, potentially on different machine.

A dealing room system is implemented based on the concept of distributed event-based programming. In a dealing room system, distributed objects reside on different machines can communicate by event subscribing and notification. The communication between objects in dealing room system is carried out by mean of java remote method invocation. One object can send notification to another object when a particular event occurs.

Events are cause and notifications are effect. The definition of event and notification are as follow.

Event: An event is a detectable condition that can trigger a notification.

Notification: A notification is an event-triggered signal sent to a run time defined recipient [8].

2. Related work

Julia Myint[4] proposed an event notification system using agent-based matchmaking services which capable of monitoring the availability of services, maintaining an updated file of all information on services use. In this system, matchmaker also matches users' interests and appropriate information automatically, and sends latest information to relevant users.

Peter R.Pietzuch[5] have introduced Hermes, a novel event-based distributed middleware architecture that follows a type- and attribute-based publish/subscribe model. It focus on the notion of an

event type and supports features commonly known from object-oriented languages like type hierarchies and supertype subscriptions.

Ludger Fiege, Gero Muhl, and Felix C. Gartner [3] have presented about the modular event based system in which the modular design and implementation of an event system is presented which supports scopes and event mappings

3. Characteristics of distributed system

The characteristics of distributed system are

Multiple Computers: More than one physical computer, each consisting of CPUs, local memory, and possibly stable storage, and I/O paths to connect it with the environment.

Interconnections: Mechanisms for communicating with other nodes via a network.

Shared State: If a subset of nodes cooperates to provide a service, a shared state is maintained by these nodes. The shared state is distributed or replicated among the participants [2].

4. Distributed objects

Distributed system requires entities which reside in different address spaces potentially on different machines to communicate. Distributed object means that objects reside in separate address spaces and whose methods can be access on remote address space potentially on different machines. The remote method call is issue in an address space separated from the address space where the target object resides. The code issuing the call is refers to as client. The target object is referred to as server object or remote object. The set of method which implement one of the server object's interfaces is sometimes designated as a service that this object provides. The process in which server object is located is referred to as server. Most importantly a method invocation on a remote object has the same syntax as a method invocation on a local object.

In the dealing room system, there are distributed objects whose methods can be invoked from another java virtual machine. They are called remote objects.

The remote objects in the dealing room system are

1. LoginClientModule
2. StockControlClientModule
3. ServerModule

5. Remote object requirements

The remote object must implements the remote interface. The remote interface is the interface that declares the methods of the remote object which can be invoked from another remote object.

1. A remote object's interface must be written as extending the java.rmi.Remote interface. This serves to mark remote objects for the RMI system. No methods are introduced by java.rmi.Remote.
2. A remote object's interface must be public.
3. A remote object's interface should extend java.rmi.server. UnicastRemoteObject. This serve to replace several object class methods so that they work properly in a distributed environment. Essentially, precautions are taken so that each client receives the same result when calling certain remote object methods.
4. All methods must be declared as throwing java.rmi.RemoteException. This could be seen as an RMI drawback - existing Java interfaces must be modified in order to function in a distributed environment.
5. Register the object using the java.rmi.Naming interface (implemented in the object's code). Note the RMI registry must be running before the object can be launched to register itself.
6. Once the object's interface is defined and an implementation is derived, the object is compiled into bytecode using the javac compiler. A client and server stub is then created from the bytecode using the RMI stub compiler, rmic. The client stub serves to provide hooks into the object serialization subsystem in RMI for marshaling method parameters.
7. The RMI registry must be running on the server. The RMI registry is launched with the command rmiregistry [PORT NUMBER].

The remote interfaces in the dealing room system are IClientModule and IServerModule. The LoginClientModule and StockControlClientModule implement the IClientModule and ServerModule implements IServerModule.

6. Events in the dealing room

Distributed event-based system extends the local event model by allowing multiple objects at different location to be notified of events taking place at an object. In a dealing room system, many events occurred between the interaction of client, server and information provider. For the occurring of a particular event, an appropriate notification is issue to relevant remote object.

A software system is said to be event-based if its parts interact primarily using event notifications. In this context, a part is anything containing code, such as a module of functions, an object, or a component made up by classes and objects. Notifications are basically signals sent from one part to another, in response to an event [8].

An event based system is made up of a collection of independent parts that interacts using event notification. A system that was designed based on

event is easier to build, test and maintain them a traditional one. The larger the system, the greater the benefits of event based approach. Event based system also tries to reduce the coupling in a system as much as possible.

6.1. Characteristics of event-based system

There are two main characteristics in distributed event-based system. They are –

Heterogeneous – When event notification are used as a means of communication between distributed object, components in a distributed system that were not designed to interoperate can be made to work together. All that is required is that event generating objects publish the types of events they offer, and that other objects subscribe to events and provide an interface for receiving notifications.

Asynchronous – Notification are sent asynchronously by event generating objects to all the objects that have subscribe to them to prevent publisher needing to synchronize with subscriber [1].

6.2. Events between client and server

The events that occur between the client and server are

1. Register
2. Login
3. Registering the interested stock
4. Canceling the registration of interested stock

Register event occurs when the new user registers to the dealing room.

Login event occurs when the registered user login to the dealing room.

Registering the interested stock occurs when the users select their interested stock which were provided by the dealing room and register with the server in order to get the notification about the changes on the price of stock.

Canceling the registration occurs when the user cancels the registration of interested stock because the user may not wish to get the notification in the future.

6.3. Events between information provider and server

The types of events that occur between the interaction of information provider and server are

1. Login
2. Insert new
3. Delete existing stock
4. Update price
5. Delete the user registration

Login event occurs when the information provider login to dealing room system.

Insert new event occurs when the information provider inserts new stock to the database on server.

Delete event occurs when the information provider deletes the stock from the database.

Update price event occurs when the information provider changes the price of stock on server with the latest price.

Delete the user registration occurs when the information provider deletes the registration of user interested stock on server.

7. Notifications in dealing room system

The main concept of Dealing Room System is to notify the user when a particular event occurs at the server. Notifications act a communication between components in this system. Client can also notify server after choosing their interested stock. In the dealing room system, the process of sending notification is handled by the server. Notifications are sent based on the occurrences of event happening at the client site and information provider site. Notifications in this system are displayed as a message box with the information of the type of event occurs.

There are two types of notification that the information provider can get about the occurrence of event in the client site. The events that cause these notifications are registering the interested stock and canceling the registration of interested stock. When the users select their interested stock and register for the event of any update occurring on that stock, server stores the user name and stock in database and sends the notification to information provider site. If so, information provider can see which user is interested on which stock and can send notification if any changes occur on that stock. When the user may not longer interested on that stock, user can cancel the registration. In such event, server deletes the registration in database and sends notification to information provider.

Client will also receive the notifications of the occurrences of events at the information provider. Whenever the information provider inserts new stock, updates the price of stock and deletes the stock that the user is interested, server sends the notification to the user with the information of event.

8. Java RMI in dealing room system

Remote method invocation (RMI) allows objects to invoke methods on remote objects. The calling objects can user the same syntax as for the local invocations. The remote method invocation (RMI) features of java enable a program running on one machine to call methods of another object created by another program running on a remote machine. The second object created by the program running on the

remote system is called a remote object. The program which creates the remote object whose methods is to be invoked is called a server. Similarly the program which invokes the remote objects methods is called the client. With RMI, programmers can perform distributed computing in a networked environment. The essence of object oriented programming is to create objects which will perform a specific task in a way transparent to other objects.

8.1 RMI between distributed objects

This system consists of separated programs which are located on different machines. They are server, client and the information provider. The remote objects are resided on these three sites. These sites interconnect by message sending which is actually an invocation of method on remote object on another site. To allow such type of communication, each site must create remote objects, makes references to those objects accessible and then waits for invocation of method on that objects.

The RMI model provides a distributed object application that the server and the client use to communicate and pass information between each other. A distributed object application has to handle the following properties.

Locate remote object: The system has to obtain references to remote objects. This can be done in two ways. Either by using RMI's naming facility, the rmiregistry, or by passing and returning remote objects.

Communicate with remote objects: The programmer does not have to handle communication between the remote objects since this is handled by the RMI system. The remote communication looks like an ordinary method invocation for the programmer.

Lode class bytecodes for objects that are passes as parameters or return values: All mechanisms for loading an object's code and transmitting data is provided by the RMI system.

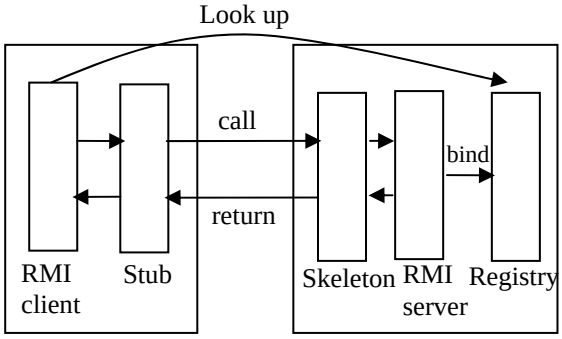

Figure 1.RMI architecture

8.2 Communication step in RMI

The communication steps in RMI are as follow.

1. Caller calls a local procedure implemented by the stub.
2. Stub marshals call type and input arguments into a request message.
3. Client stub sends the message over the network to the server.
4. Server skeleton receives the request message from the network.
5. Skeleton unpacks call type from the request message and looks up the procedure on the called object.
6. Skeleton unmarshalls procedure arguments.
7. Skeleton executes the procedure on the called object.
8. Called object performs a computation and returns the result.
9. Skeleton packs the output arguments into a response message.
10. Skeleton sends the message over the network back to the client.
11. Client stub receives the response message from the network.
12. Stub unpacks output arguments from the message.
13. Stub passes output arguments to the caller [9].

9. System flow diagrams

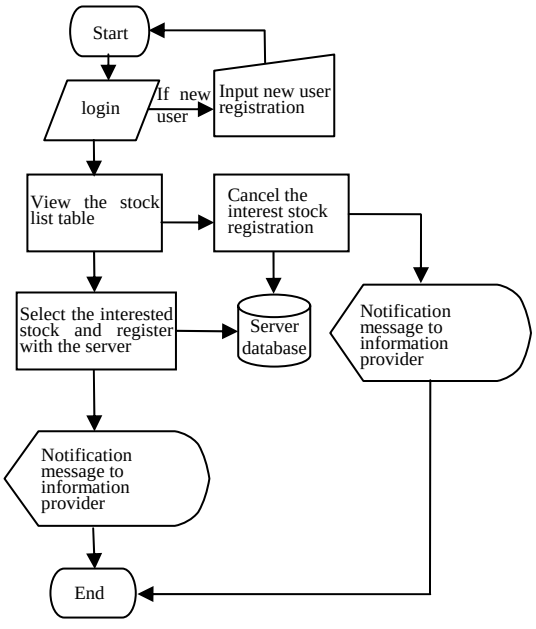

Figure 2.System flow of client and server sites

In this system, client and server are located on different machines. If the client login to system, the user name and password of client is validated with the information stored at server. If the login information is valid, server returns the correct information and client can view the stock list which stored at the server database. If the particular event occurs at the client site, server will send notification message to information provider which resides on other machine.

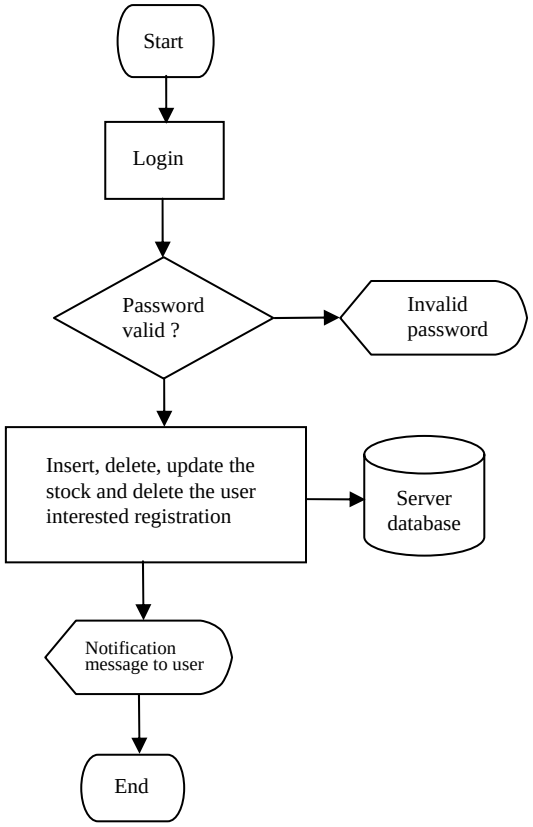

Figure 3.System flow of information provider site

Information provider must login to the system with correct name and password. Information provider is authorized to access the information stored at the server site. When the provider performs changes on stock in database, server will send notification to appropriate users who register interested on that stock.

10. Implementation of dealing room

Figure 4.Stock control form

Information provider can add new stock, change stock price and delete the existing stock. To insert new stock, the provider must type the id number, name of stock, date and price of stock. To delete the stock, provider must choose id of stock. By pressing “Delete by id” button, the stock with selected id will be delete from the list. To update the existing stock, provider must choose the id of stock, enter new information and press “Update stocks” button. The stock with selected id will change. The “Update by interested stock” and “Delete by interested stock” buttons are used to update the stock which is registered by the user who interested on that stock.

Figure 5.Stock list table

When the user login to the dealing room, the dealing room will show stock table. User can see the stock prices and the latest updates of stock. If the user wants the notification about changes in a particular stock, user can choose the particular stock id. The server will save user name and stock name when “stock register” button was pressed. Then user will get the notification about any updates on that stock. If user does not want notifications anymore, user can cancel the stock registration.

11. Conclusion

Event oriented model was introduced in distributed system since 1970s [8]. Many application and operating system such as Microsoft Windows use graphical user interface based on event. Dealing room system was designed based on the aspect of event and notification. This approach is simple and easy to develop by using Java Remote Method Invocation (RMI). This system and provide the present price of stock on market. When the changes on price occur, it can inform to user immediately by sending notification massage. Because it was based on the event and notification paradigm, user does not need to couple with the system and the communication between the system and users is asynchronous.

12. Limitations

The connection between components in the system is peer to peer connection. Server marks each user with their IP address. So that each client machine in the system can be used by a particular user.

13. References

[1] G.Coulouris, J.Dollimore and T.Kindberge, *Distributed system Concept and Design Third Edition*. Pearson Education Press, 2001.

[2] F.Jahanian, *Introduction to Distributed Systems*. http://www.eecs.umich.edu/~farnam.

[3] Ludger Fiege, Gero Muhl, and Felix C. Gartner, *Modular Event-Based Systems*, Darmstadt University of Technology D-64283 Darmstadt, Germany.

[4] Julia Myint, *Event Notification System Using Agent-Based Matchmaking Services.*

[5] Peter R.Pietzuch, *Hermes: A Distributed Event-Based Middleware Architecture.*

[6] F.Plasil and M.Stal, *An architectural view of distributed objects and components in CORBA, Java RMI and COM/DCOM*, Springer-Verlag, 1998.

[7] T.Sun& A.´E Trudel, *An implemented e-commerce shopping system which makes personal recommendations*, Jodrey School of Computer Science Acadia University Wolfville, Nova Scotia, B0P 1X0 Canada.

[8]T.Faison and F.De Gasperis, *Event-Based programming: Taking events to the limit*, ISBN-13: 978-1-59059-643-2, 2006.

[9]http://www.wikipedia.com/Distributed object communication. PDF