Developments in Business Simulation and Experiential Exercises, Volume 26, 1999

FINANCIAL SIMULATION USING DISTRIBUTED COMPUTING TECHNOLOGY

Frank Duserick, Alfred University
Ernest Enke, Alfred University
Wilford Huang, Alfred University
Abderahman Robana, Alfred University

ABSTRACT

This paper describes a Java™ based and Internet-based simulation prototype for financial decision making. Java™ was selected for software development because of the ease with which one could write code for complicated purposes and because of the security it provides while running across different platforms. The simulation game Robastrat was developed to show how distributed computing technology helps in building efficient and effective user-friendly simulation applications using Java™.

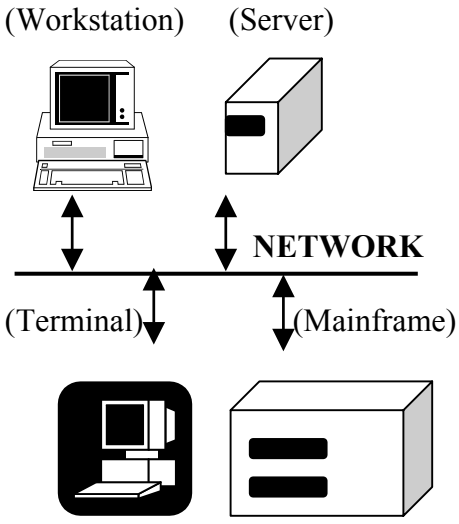
INTRODUCTION

This paper describes the efficacy of using Java™ to develop an internet-based simulation for financial decision making. Historically, financial decision making simulation packages have been platform dependent resulting in the need to design multiple platform specific modules for the same simulation. The simulation game, Robastrat, is designed to show how distributed computing technology helps in building efficient and effective user-friendly applications using Java™. The design and implementation of this simulation required in-depth knowledge of relational data base theory, of distributed computing, of the Java object-oriented programming language, and of Java™ Remote Method Invocation (RMI). Use of Robastrat requires the user (student) to have basic knowledge of the internet. Java™ RMI is the method used to address this web-based client-server simulation. A key assumption of this research is that the Internet will be a universal medium for communication, Comer (1993).

DATABASE APPLICATIONS

By definition a database is a collection of data of any kind or any size. Databases have become an integral part of everyone's daily curriculum regardless of whether it is the field of engineering or medicine or computer science. Storage and preservation of data has been and still is a crucial issue for any large application. Day-to-day innovations in the field of client/server technology (figure 1) are becoming more and more common and increasingly competitive. The transition from terminal/mainframe environment to desktop PCs and servers has become popular in the realm of Business and Management Information Systems. Transferring files (data) between clients and allowing users to run programs on other remote systems are integral aspects of most organizations' networking strategies. The database is now the underlying framework of information systems, and has fundamentally changed the way many organizations operate, Connolly et al (1999).

FIGURE 1
CONTEMPORARY CLIENT/SERVER ARCHITECTURE



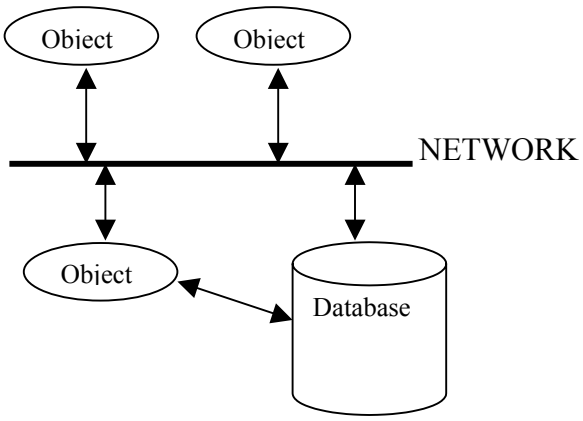
In the late 1980s, LANs formed the framework for connecting separated microcomputers enabling them to send information at very high rates. The LAN-based multi-user structure is considerably different from the multi-user structure used on mainframe databases. The job of processing the data or entertaining the requests of an end user is shared among the different CPUs involved in the LAN system. Eventually the idea of client-server architecture came of age, allowing transfer of multimedia applications via Internet and Intranets. The concept of 'fat-client' won the trust of most computer users and emerged as a revolution that made data access and sharing information, much more flexible and user-friendly than ever. Today, most end-user devices are termed 'fat-clients' meaning standalone workstations that consist of proprietary hardware and operating system with client-based applications.

DISTRIBUTED COMPUTING

There exists a distinct transition in the way businesses utilize Management Information Systems for their welfare. Remote access and management has always been in the picture, working its way through alongside web-based database applications. The client/server computing model defines the way successful organizations will use technology during the next decade. Enabling technologies such as object-oriented development and graphical user interfaces (GUIs) will liberate the users of information to use technology personally and directly, Smith (1992).

A client is any single-user standalone PC that has the necessary connectivity with the required host (server) and is capable of performing the computing needed to interact with the database on the host, through the designed GUI. A server is usually one or more multi-user processors with shared memory; it provides the required connectivity and service to all the dependent clients through various available techniques.
Currently, Distributed Object Computing (DOC) has gained popularity as a vehicle for constructing and maintaining network-centric applications. Distributed computing has the unique feature of making use of remote objects that may be placed at an isolated place anywhere on the network (figure 2). Java™ being platform-independent adds more flexibility and sophistication in developing web-based applications. While Common Gateway Interface{CGI} and Application Programming Interface (API) represent the most popular distributed computing techniques, Horton (1997), these methods represent the end points of a communication link similar to any telephone connectivity. The Robastrat simulation program developed shows that Java™ RMI surpasses the limitations of CGI and AGI.

FIGURE 2
DISTRIBUTED COMPUTING ARCHITECTURE



REMOTE METHOD INVOCATION (RMI)

RMI is a well-structured Object Request Broker (ORB) which allows the transfer of code in addition to data. It utilizes Java as an interface definition language as well as implementation language. RMI takes full responsibility of the security and legal standards of the code that would be transferred to the client site. URL-based naming scheme and ability to pass objects over the network are innovations that bring out a novel way for client/server applications.
While RMI does have some failings, they are trivial when compared to the power it grants for

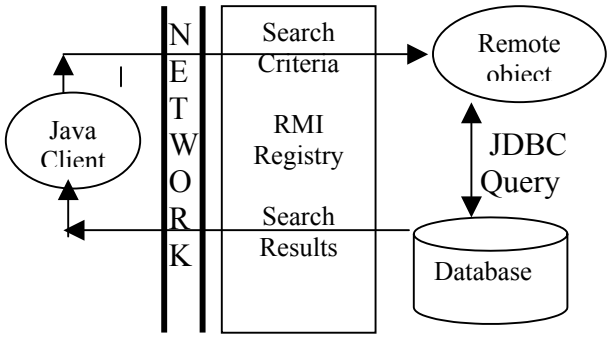
deploying robust applications. It is indeed dependent on Java and requires only Java programming language to be used to make the most of it. It is not as fast as raw sockets when it comes to data transfer on the network. Every time a new remote object is added to the existing software code, the RMI registry has to be restarted again. In Robastrat, the browser with which the client wants to download should have the latest Java plug-in and the Plug-in HTML converter so as to ensure hassle free deployment of the applet. Nevertheless, Java RMI appears to be a robust solution for better design of client/server applications.

Remote Method Invocation Java™ (RMI) provides easy-to-use Java-based distributed computing facilities. It is a set of classes and interfaces that permit the usage of the public methods belonging to any object residing on the same machine or at a remote location. Unlike CGI, RMI allows Java™ programs to call certain registered methods on a remote server and does not compel the client or the server to run on an HTTP server.

RMI can provide a tenacious connectivity for the client to access the remote server. Specifically, the client's application is entertained until the access to the server is recalled by the application itself. Contrary to the three-tier system having a middle-ware (e.g. JDBC™) to handle the important functions, RMI doesn't require individual clients to have the database drivers. The client machine is independent of the database characteristics and hence immune to any changes occurring to the database or the database engine.

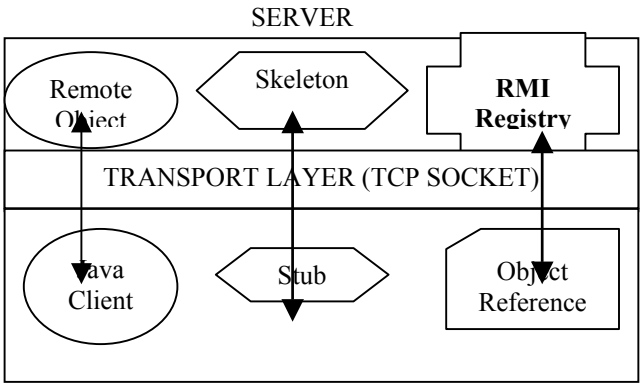
Java™ RMI provides easy-to-use Java-based distributed computing facilities. It brings a new level of functionality to distributed programs with features like distributed, automatic management of objects and passing objects themselves from machine to machine over the network. It enables developers of distributed Java™ applications to treat remote objects and their methods very much like normal Java objects. Java™ RMI is a set of APIs and a model for remote objects that allows developers to build distributed applications very easily in Java™. It uses normal Java™ interfaces to define remote objects rather than a separate language like IDL. Currently Java™ RMI uses a combination of Object serialization and the Java Remote Method Protocol (JRMP) to turn normal looking method invocations into remote method invocations, Van Hoff (1996).

FIGURE 3
JAVA RMI ENVIRONMENT



In distributed computing, an object calls procedures contained within another object and the second object in turn may call the procedures from yet another object. The name 'Remote Method Invocation' itself explains the principle within. RMI involves a client invoking a remote method of a remote object, resident on a remote machine, physically separate but logically conversing through the remote interface (figure 3). The Remote server comprises one or more remote objects that work in unison with the server side application, RMI registry.

FIGURE 4
STUB-SKELETON CONFIGURATION



Each remote object should register itself with the RMI registry, which then would maintain all references to these remote objects. Should the client application need to use one or more methods of the remote object, it would request the RMI registry first to ensure the registration of the called object with the RMI registry. Upon confirmation, the RMI registry would issue a reference to that object called the 'Stub', with it's counter part called the 'Skeleton' on the server side (figure 4). Instead of the client application directly making use of the remote object, it makes use of the methods in the stub and the stub transfers the arguments to the skeleton, which in turn passes the arguments to the actual object's method. The processed results are returned back to the skeleton, then to the stub, and finally to back to the client. The exchange of information between the client and the server objects is performed through object serialization.

ROBOSTRAT SIMULATION

The Simulation

RobaStrat is an interactive portfolio strategy simulation designed to help understand the stock market better. Users of this simulation are termed 'Investors' and the Investors are provided with an uncertain environment in which to make strategic decisions aimed at maximizing their wealth, Gitman et al (1981), Law & Kelton (1982), and Naylor et al (1981). Wealth is defined as assets minus liabilities, Brigham & Houston (1996).

The administrator will have the choice of using the stock market data downloaded directly from the market source at real time, or random in-house data replicating the conditions of bull and bear markets in different degrees of intensity of price movements. The random in-house data option, the administrator will have the opportunity to trigger bull or bear conditions. It is recommended that initially the students (investors) should use normal buy and sell transactions. Students are provided with two types of information at a cost one less than the other, as it provides less information than the other about the probable market price condition and the price execution. Basically, prices are affected by the supply and demand conditions, and a random factor reflecting the capriciousness of the stock market and market psychology.

The Implementation of the Simulation

Distributed Object Computing (DOC) provides the unique capability to design and develop network-centric applications without the hassles of the traditional tiered architecture. The approach of RMI was implemented in designing an n-tier database application, written purely in Java™. The application is a Java™ applet that can be downloaded from an HTTP web site to the client's site, equipped with Netscape 4.05 as the browser.

The usual bottlenecks that are encountered during the design of a client/server application are enumerated as follows, Harold(1997):

1. The network traffic is a very crucial aspect that affects the speed of data transfer over the Internet and the loss or delay of data is very annoying. The distance between the client and the server counts as an important issue when uninterrupted data access and collision free domains become rare.
2. The number of tiers through which the query of the client passes, and the number of tiers the result set of the inquiry goes through, is considered as a critical issue.
3. The security that promises restricted access to the users of the database and not to create accidental formatting of local hard drive of deletion of files from the local file systems is necessary.
4. Compatibility of the user's computer system, whether the application would be able to run on the client's operating system is again an aspect that is to be considered. Not every application is universally accepted by all operating systems.
5. Since the client is presumed to be at a remote location, the application has got to be web-based and should be made available through a web-server.

6. Client should be granted the freedom of having very few requirements for running the application on his own machine at his own place.

All such factors are to be considered when an application is to be developed that suits the needs and choices of a variety of clients. The combination of Java™ and JDBC™ provides the appropriate integration of cross-platform dexterity and the ubiquity of write once and run it anywhere Horton(1997). JDBC™ makes it possible to send SQL statements to virtually any relational database.

RMI was utilized to handle the issue of distributed database applications. There is not much difference between the JDBC™ way and the RMI way. The client, the server and the database are still physically separate (on different virtual machines) but are logically linked through the TCP socket. Since in RMI each client object can call the public methods of any remote object, the caller is considered a client and called object is considered a server. Therefore, in a network you end up with an n-tier system with many callers and called objects, instead of a three-tier system. Of all the remote objects available on the server side, only one talks to the database and does all the necessary transactions. Besides the flexibility of managing the database centrally, there is no need for the native database driver to be installed on each of the client machine as in the case of tiered architecture. Probability of collisions because of multithreading are very much reduced and the network traffic is more stable. At any time, other applications that need to make use of the same remote object for the same kind of data accessing logic, are free to use it instead of rebuilding the entire code for a second time.

In the Robastrat simulation program the Java application (applet) is constructed and is resident on a Windows 95/98 system. The RMI registry is run on the same machine that then forms the host that serves multiple clients at the same time. The principle of multi-threading is used to serve 'n' number of clients. Since the application is programmed with Java programming language, the client may have a system with UNIX, OS/2, Windows 95, or Windows 98. In our implementation the server (host) itself is the web server that hosts the web page from which the applet can be downloaded and the database corresponding to this application also dwells on the same machine.

Security is a very crucial issue especially when dealing with client/server computing. All the required security measures are to be taken in order to ensure hassle free distributed environment. If the client application were an applet then the applet itself would have its own security manager to take care of the illegal access to local file systems. If the client application is a regular application other than applets, then the security manager need to be constructed and deployed, Hamilton et al (1997).

CONCLUSION

Despite the various techniques available to make web pages more interactive and user-friendly, and to make database applications on the network more flexible, RMI has proven to make the tasks much faster and easier than ever. With RMI in use, the requirements for the client are drastically reduced. Remote method Invocation-a great solution for distributed computing, put to the best use, helps making applications like RobaStrat. This study proves to be a wonderful model, especially for students who want to master the state-of-art in the area of distributed computing.

REFERENCES

Brigham, Eugene F. & Houston, Joel F. (1996). *Fundamentals of Financial Management-The Concise Edition*. Fort Worth, Texas: Dryden Press.

Comer, Douglas E. (1994). *The Internet Book-Everything you need to know about computer networking and how the Internet works.* New Jersey: Prentice Hall

Connolly, Thomas, Begg, Carolyn & Strachan, Anne (1999). *Database Systems - A Practical Approach to Design, Implementation, and Management*. Harlow, England: Addison-Wesley, Second edition

Gitman, Lawrence J. Robana, Abderrahman and Biggs, William D (1981) . *Portstrat - A Portfolio Strategy Simulation*. New York: John Wiley & Sons

Hamilton, Graham, Cattell, Rick & Fisher, Maydene (1997). *JDBC Database Access with Java - A Tutorial and Annotated Reference*; Ch 1, 2, 3. Reading, Massachusetts: Addison-Wesley,

Harold, Elliotte Rusty (1997*). Java Network Programming*; Ch 14. Sebastopol, California: O'Reilly

Horton, Ivor (1997). *Beginning Java*; Ch 19. Olton , Birmingham: Wrox Press

Law, Averill M. & Kelton, W. David (1982). *Simulation Modeling & Analysis* Second Edition. New York: McGraw-Hill,

Naylor, T.J.,.Balintfy, J.L. Burdick, D.S & Chu, K. 1981. *Computer Simulation Techniques.* New York: John Wiley & Sons,1966 as cited by Reuven Y. Rubinstein, Simulation and the Monte Carlo Method.

Smith, Patrick (1992). *Client/Server Computing-Professional Reference Series*; Ch 2, 5, 10. Carmel, Indiana: Sams Publishing

Van Hoff, Arthur, Shaio, Sami & Starbuck, Orca (1996*). Hooked on Java-Creating Hot Web Sites with Java Applets*; Ch 1, 2. Reading, Massachusetts: Addison-Wesley Developers Press