iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 3 **Issue:** V **Month of publication:** May 2015

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Distributed Java Applications

Meenakshi[#1], Sanjiv Kumar Singh[#2]
M.Tech, Assistant Professor,
Department of CSE, Shri Balwant College of Engineering &Technology, DCRUST University

Abstract— This paper presents a new Java oriented monitoring infrastructure that enables tools to observe, analyze and manipulate the execution of distributed Java applications independent of implementation details like instrumentation of monitored entities, hardware platform and application libraries. Tools can access the monitored application via a standardized interface defined by an On-Line Monitoring Interface Specification (OMIS) and extended by a set of new Java-specific services relating to garbage collection, class loading, remote method invocation, etc. The new monitoring functionality can be applied for building various kinds of tools and for adapting the already existing ones, such as performance analyzers, debuggers, etc., working in the on-line mode.
Keywords— Java, monitoring system, monitoring interface, distributed object system, OMIS

I. INTRODUCTION

Java, as a relatively simple, object-oriented, secure and portable language, is also a flexible and powerful programming system for distributed computing. Program development with Java results in software that is portable across multiple machine architectures and operating systems. Distributed programming in Java is supported by remote method invocation (RMI), object serialization, reflection, a Java security manager and distributed garbage collection. Java RMI is designed to simplify the communication between objects in different virtual machines allowing transparent calls to methods in remote virtual machines.

A major disadvantage of Java is the speed of execution, especially in the case of distributed applications, due to the limited bandwidth of the communication channel and the added delay caused by the JVM translating the byte code and garbage collection. A developer of distributed systems faces various problems that make developing such systems more difficult than expected. In a complex distributed application, performance optimization of the code becomes much more important and requires programmers' close attention. Understanding the nature of Java-related problems allows programmers to build large and scalable applications. However, without a suitable performance analysis tool for Java programs, it is often difficult to analyze a program for performance-tuning and bug detection. Thus, there is a need for tools (performance analyzers, debuggers, etc.) that allow programmers to control and improve their applications.

In this paper, we focus on the issues of building a monitoring platform of the above properties based on a well-defined interface between a monitoring system organized as middleware and tools that use the monitoring facilities provided by the monitoring middleware.

II. LITERATURE REVIEW

This section gives a review of Handwritten Character The first version of JVMs had poor support for monitoring Java programs. Initially there was a simple debugger, jde, attached to the Java Development Kit (JDK). Then, there was an instrumented Java virtual machine build for JDK version 1.16 to support the collection of profiling data generated when executing a Java program. This approach was developed until version 2 of the Java platform. All JVMs for the new Java platform were equipped with interfaces for debugging (JVMDI) [6] and profiling (JVMPI) [7]. A new release of Java 2 Platform version 1.5, called Tiger, contains a new native profiling interface called JVMTI which is intended to replace JVMPI and JVMDI. JVMTI aims to cover the full range of native in-process tools access, which in addition to profiling, includes monitoring, debugging and, potentially, a wide variety of other code analysis tools. Additionally, Tiger's implementation includes a mechanism for bytecode instrumentation, the Java Programming Language Instrumentation Service (JPLIS). This enables performance analysis tools to execute additional profiling only where needed. The advantage of this technique is that it allows for more focused analysis and limits the interference of the profiling tools in a running JVM. The instrumentation can even be dynamically generated at runtime, as well as at class loading time, and pre-processed as class files.

Most of the tools for JVM versions from 1.2 to 1.4 are based on the Java Virtual Machine Profiling Interface (JVMPI) [7]. Starting with JDK 1.2 SDK it also includes an example profiler agent for efficiency examination called hprof [8], which can be used to build professional profilers. A Heap Analysis Tool (Hat) [9] enables one to read and analyze profile reports of the heap generated by the hprof tool and may be used e.g. for debugging "memory leaks". Tracer [10] is a debugger which provides traditional features, e.g. a variable watcher, breakpoints and line-by-line execution. J-Sprint [11] provides information about what parts of a program consume the most of execution time and memory. JProfiler [12], targeted at JEE and JSE applications,

©IJRASET 2015: All Rights are Reserved

provides information on CPU and memory usage, thread profiling and VM. Its visualization tool shows the object references chain, execution control flow, thread hierarchy and general information about JVM using special displays. There is also a group of powerful commercial tools with friendly graphical interfaces: OptimizeIt [13], Jtest [14] and JProbe [15, 16], which enable identification of performance bottlenecks.

All these tools have similar features: memory, performance, code coverage analysis, program debugging, thread deadlock detection and class instrumentation, but many of them are designed to observe a single-process Java application and do not support directly monitoring a distributed environment based on RMI middleware, except for JaViz [17], which is intended to supplement the existing performance analysis tools with tracing client/server activities to extend Java's profiling support for distributed environments.

The above mentioned tools provide a wide range of advanced functionalities, but practically each of them only provides a subset of the desired functions. Distributed systems are very complex and the best monitoring of such a system could be achieved by using diverse observation/manipulation techniques and mechanisms. It is therefore often desirable to have a suite of specialized tools such as debuggers and performance analyzers, each of them addressing a different support issue and allowing developers to explore the program's behavior from various viewpoints. Therefore, there is a need to establish a more general approach to build flexible, portable and efficient monitoring based tools.

III. DISTRIBUTED APPLICATIONS

The term *distributed applications,* is used for applications that require two or more autonomous computers or processes to cooperate in order to run them. Thus, the distributed system considered in this thesis, involves three resources, processing, data and user interface. Both processing and data can be distributed over many computers. The user interface is usually local to the user so that the graphical interface, which consumes high bandwidth, does not have to be transmitted from one location to another (figure 3.1).

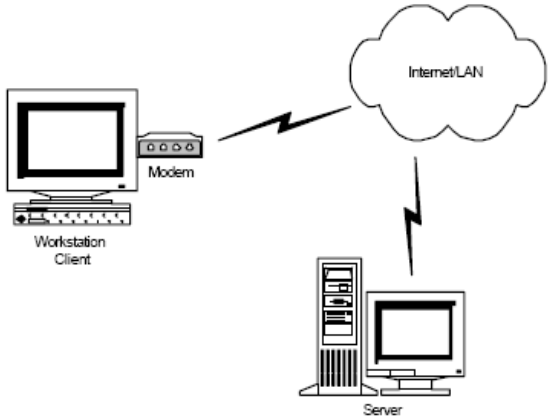


Fig. 3.1: A Typical Distributed Application Scenario

In distributed computing, the computer network is used to support the execution of program units, called processes that cooperate with one another to work towards a common goal. This approach has become popular due to a number of developments like:

Increase in the number of personal computers

Low cost of establishing computer networks with the advancement of technology

Computer manufacturers now offer networking software as a part of the basic operating system

Computer networks are now an established way of disseminating information

The modern client/server model uses **proxy objects** for server and client respectively. The client calls the proxy, making a regular method call. The client proxy contacts the server. Similarly, a second proxy object on the server communicates with the client proxy, and it makes regular calls to the server object.

A. Methods of proxies' communications

There are three different methods with which proxies communicate with each other.

©IJRASET 2015: All Rights are Reserved

1) RMI, the Java Remote Method Invocation technology, supports method calls between distributed Java objects.
2) CORBA, the Common Object Request Broker Architecture, supports method calls between objects of any programming language. CORBA uses the Internet Inter-ORB Protocol or IIOP to communicate between objects.
3) SOAP, the Simple Object Access Protocol, is also programming – language neutral. However,

SOAP uses an XML-based transmission format

IV. REMOTE METHOD INVOCATION

RMI allows a Java object that executes on one machine to invoke a method of the Java object that executes on another machine. This is an important feature, because it allows building distributed application. All the RMI classes are available in jave.rmi package. To use different classes of this package we must import the java.rmi package in the beginning of the Java program.
One main application where RMI is used client/server. The server receives requests from a client, processes it & returns the result. For example, the client seeking product information can query a Ware House object on the server. It calls a remote method, *find*, which has one parameter: a Customer object. The *find* method returns an object to the client: the Product

Object (Figure4.1).

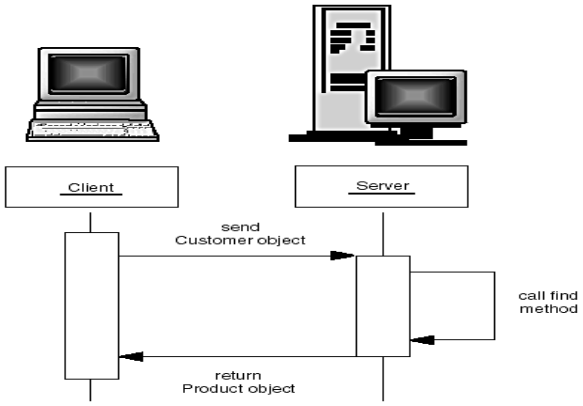


Figure 4.1: RMI using Client & Server Object

In RMI terminology, the object whose method makes the remote call is called the client object. The remote object is called the server object.

V. COMMON OBJECT REQUEST BROKER ARCHITECTURE

Though RMI is a powerful mechanism for distributing and processing objects in a platform-independent manner, it has one significant drawback. it only works with objects that have been created using Java. Convenient though it might be if Java were the only language used for creating software objects, this simply is not the case in the real world.
A more generic approach to the development of distributed systems is offered by CORBA (Common Object Request Broker Architecture), which allows objects written in a variety of programming languages to be accessed by client programs which themselves may be written in a variety of programming languages.
Another fundamental difference between RMI and CORBA is that, whereas RMI uses Java to define the interfaces for its objects, CORBA uses a special language called **Interface Definition Language (IDL)** to define those interfaces. Although this language has syntactic similarities to C++, it is not a full-blown programming language. In order for any ORB to provide access to software objects in a particular programming language, the ORB has to provide a *mapping* from the IDL to the target language. Mappings currently specified include ones for Java, C++, C, Smalltalk, COBOL and Ada.
At the client end of a CORBA interaction, there is a code **stub** for each method that is to be called remotely. This stub acts as a proxy (a 'stand-in') for the remote method. At the server end, there is **skeleton** code that also acts as a proxy for the required method and is used to translate the incoming method call and any parameters into their implementation-specific format, which is then used to invoke the method implementation on the associated object. Method invocation passes through the stub on the client side, then through the ORB and finally through the skeleton on the server side, where it is executed on the object. For a client and server using the same ORB, Figure 5.1 shows the process.

©IJRASET 2015: All Rights are Reserved

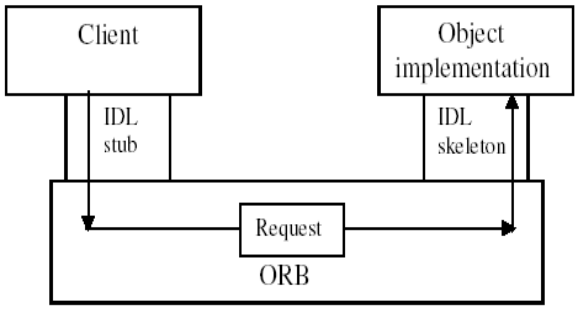


Figure 5.1: Remote method invocation when client and server are using the same ORB.

VI. SIMPLE OBJECT ACCESS PROTOCOL

IBM, Lotus Development Corporation, Microsoft, Develop Mentor and User land Software developed and drafted SOAP, which is an HTTP-XML-based protocol that enables applications to communicate over the Internet, by using XML documents called *SOAP messages*.

SOAP is compatible with any object model, because it includes only functions and capabilities that are absolutely necessary for defining a communication framework. Thus, SOAP is both platform and software independent, and any programming language can implement it. SOAP supports transport using almost any conceivable protocol. SOAP binds to HTTP and follows the HTTP request–response model.

SOAP also supports any method of encoding data, which enables SOAP-based applications to send virtually any type information (e.g., images, objects, documents, etc.) in SOAP messages. A SOAP message contains an *envelope*, which describes the content, intended recipient and processing requirements of a message. The optional ***header*** *element* of a SOAP message provides processing instructions for applications that receive the message.

VII. CONCLUSION

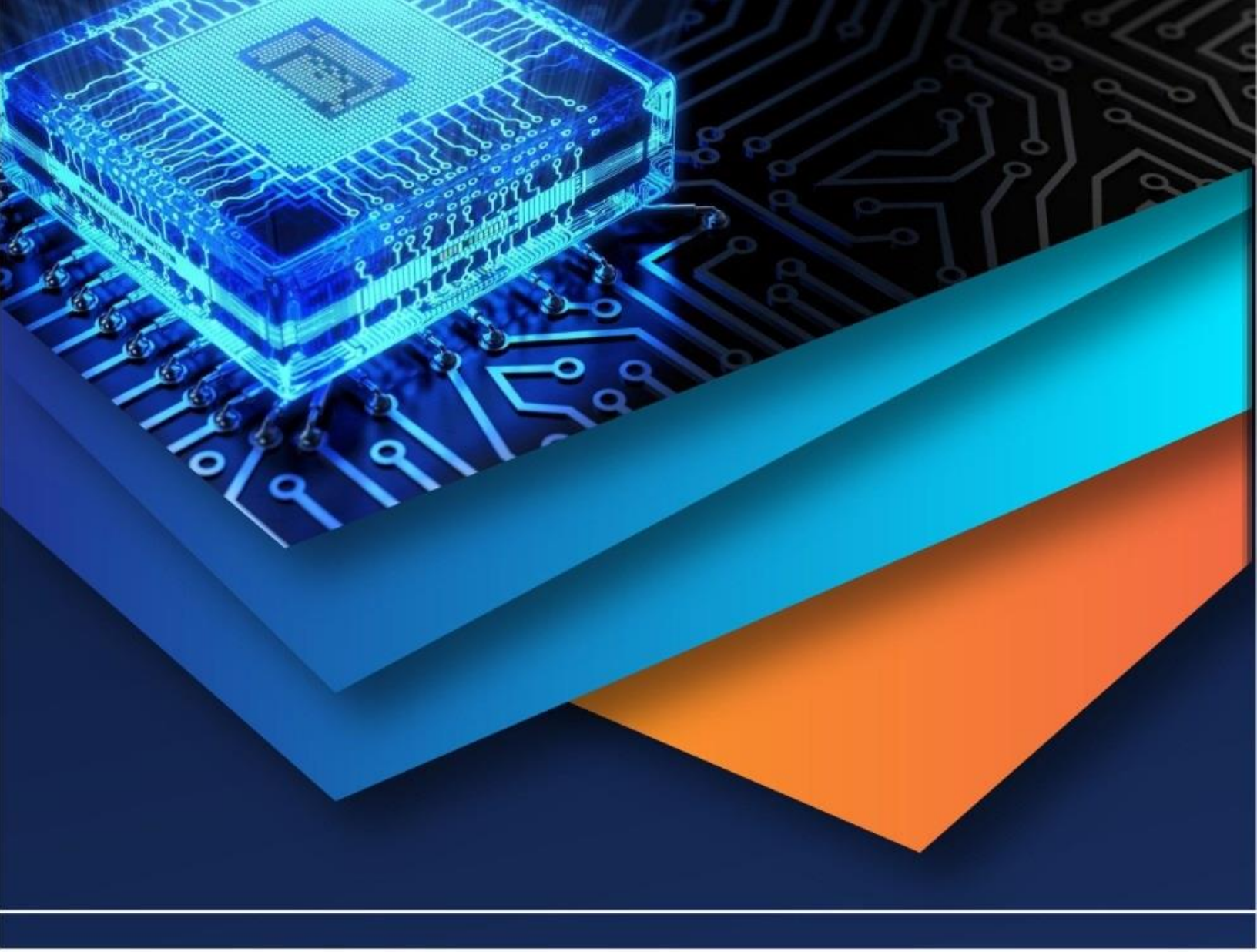
During the first two decades of their existence, computer systems were highly centralized. A computer was usually placed within a large room and the information to be processed had to be taken to it. This had two major flaws, a) the concept of a single large computer doing all the work and b) the idea of users bringing work to the computer instead of bringing the computer to the user. This was followed by 'stand alone PCs' where the complete application had to be loaded on to a single machine. Each user has his/her own copy of the software. The major problems were a) sharing information and b) redundancy.

These two concepts are now being balanced by a new concept called computer networks. In computer networking a large number of separate but interconnected computers work together. An application that requires two or more computers on the network is called a network application. The client–server model is a standard model for network applications. A server is a process that is waiting to be contacted by a client process so that the server can do something for it. A client is a process that sends a request to the server.

REFERENCES

[1] Bubak M, Funika W, Metel P, Orłowski R and Wism¨uller R 2002 Proc. 4 th Int. Conf. PPAM 2001, Naleczow, Poland, LNCS 2328 315
[2] Bubak M, Funika W, Smetek M, Kilianski Z and Wism¨uller R 2003 Proc. 10 th European PVM/MPI Users' Group Meeting, Venice, Italy, LNCS 2840 447
[3] Bubak M, Funika W, Wism¨uller R, Metel P and Orłowski R 2003 Future Generation Computer Systems 19 651
[4] Bubak M, Funika W, Smetek M, Kilianski Z and Wism¨uller R 2004 Proc. 5 th Int. Conf. PPAM, Czestochowa, Poland, LNCS 3019 352
[5] Funika W, Bubak M, Smetek M and Wism¨uller R 2004 Proc. Int. Conf. on Computational Science, Cracow, Poland, LNCS 3038 472
[6] Sun Microsystems: Java Virtual Machine Profiler Interface (JVMDI), http://java.sun.com/j2se/1.4.2/docs/guide/jpda/jvmdi-spec.html
[7] Sun Microsystems: Java Virtual Machine Profiler Interface (JVMPI), http://java.sun.com/j2se /1.4.2/docs/guide/jvmpi/jvmpi.html
[8] The SDK Profiler, http://www.javaworld.com /javaworld/jw-12-2001/jw-1207-hprof.html
[9] Sun's Heap Analysis Tool (HAT) for Analysing Output from hprof, http://java.sun.com/developer/onlineTraining/Programming/JDCBook/hat bin.zip
[10] JTracer Tool, http://amslib.free.fr/
[11] Java Profiler J-Sprint, http://www.j-sprint.com/
[12] JProbe, http://java.quest.com/jprobe/jprobe.shtml
[13] JView, http://www.devstream.com/
[14] Kazi I H, Jose D P, Ben-Hamida B, Hescott C J, Kwok C, Konstan J, Lilja D J and Yew P-C 2000 IBM Systems Journal 39 (1) 96; http://www.research.ibm.com/journal/sj/391/kazi.html
[15] Sun Microsystems: Java Platform Debug Architecture (JPDA), http://java.sun.com/j2se/1.4.2/docs/guide/jpda/

©IJRASET 2015: All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)