Tracing Facility for Visualization system of Distributed Java Object Application

o

† Dong-Woo Lee and ‡R.S. Ramakrishna

†, ‡Dept. of Information and Communications, K-JIST
1 Oryong-dong, Puk-gu, Kwangju 500-712 Republic of Korea
Email: {leepro, rsr}@kjist.ac.kr

Abstract

Distributed Object Systems are very complex. So, it is difficult to see overall relationship among objects participated in the system. Moreover the performance tuning or maintenance are also important issues of it. So, it needs a way to view the system with low-cost and an efficient method. One of solutions is a visualization tool or system. In this paper, we proposed a tracing facility for Java-based distributed object system, especially RMI(Remote Method Invocation). Our up-coming visualization system will use two phase hybrid post-mortem/on-the-fly technique. To support it, the fundamental tracing part must have some flexible and dynamic mechanism. The main idea of our tracing technique is the Plug-in Sensor Model(PSM). The relationship between tracing (monitoring) part and visualization part is closely related. So, we considered the appropriate factors for visualization. We developed 'Traced RMI(TRMI)'. For more precise visualization of a working system, the casuality of events has to be preserved. TRMI can support global event ordering.

KEY WORDS : *Trace, Java, Remote Method Invocation, Distributed Computing, Object, Visualization*

1 Introduction

Understanding and analyzing the execution of a distributed application can be a difficult task. Activities associated with distributed computing that must be problematic include program understanding, debugging, and performance tuning. One approach to addressing the complexities related to distributed computing is the use of visualization. Program and/or performance visualization can serve as an aid for all of the above problematic activities associated with distributed computing [2] These days, Many distributed systems are constructed at many sites for their own purpose. Moreover, these systems are mostly based on the Distributed Object-based system such as CORBA, Java RMI, EJB and DCOM. Recently, Java RMI or JavaBeans which is the standard Java Object Component is applied to many kinds of distributed applications. It is difficult to see the overall relationship among distributed Java object. And also there is no way to find out the bottleneck of transactions of object communication. So, it needs a tool for viewing the overall system visually. We can tune the system corresponding to the graphical presentation that is produced by the visualization system. We can obtain the better performance of the system and the way for maintenance[1].

To Visualize a distributed object system, we need a way to extract information related to the execution system. That is the instrumentation or tracing system which intercept operations taken place in the traced system to extract useful information for visualization. There are some intrusive overhead that is produced when the tracing facility is working. There are several visualization systems available such as Distirbuted Object Visualization Environment(DOVE)[3], Automated Instrumentation and Minitoring System(AIMS)[4], Pablo[5] and Falcon[6]. These researches seek ways to aid a distributed system for understanding, development, testing, and debugging by using visualization. Since they are specialized in specific systems, they can not support a distributed object system like CORBA or Java. So, we designed and implemented tracing facility for Distributed Java Object Application based on Java RMI(Remote Method Invocation).

In section 2, we introduce Java RMI briefly. Section 3 contains a detailed description of the plug-in

sensor model for tracing object interactions. In section 4 we show implementations. Section 5 summaries our experience and describes future directions.

2 Distributed Objects and RMI

Distributed objects are a potentially powerful tool that has only become broadly available for developers at large in the past few years. The power of distributed objects lies in that any software agent in our system can directly interact with an object that 'lives' on a remote host. Distributed objects really give you a tool for opening up your distributed system's resources across the board like Figure 1.

The Java Remote Method Invocation(RMI) is a Java-centric scheme for distributed objects that is now a part of the core Java API[7]. RMI offers some of the critical elements of a distributed object system for Java, plus some other features that are made possible by the fact that RMI is a Java-only system. RMI has object communication facilities that are analogous to CORBA's IIOP,and its object serialization system provides a way for you to transfer or request an object instance by value from one remote process to another. Since RMI is a Java-only distributed object scheme,all object interfaces are written in Java. Client stubs and server skeletons are generated from `java.rmi.Remote` interface. Once the remote objet's Java interface is defined, a server object implementation of the interface can be written. The server object extends the `java.rmi.server.UnicastRemoteObject` class.

The `rmic` compiler bootstraps off of the Java bytescodes for the object interface and implementation to generate a client stub and a server skeleton for the class. In RMI, the `rmiregistry` serves the role of Object Manager and Naming Service for the distributed object system.

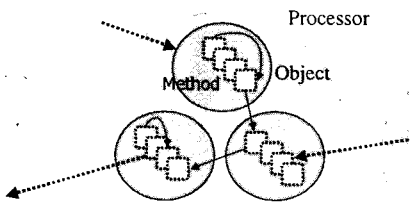

Figure 1: Distributed Object System

3 Plug-in Sensor Model

To trace interactions among remote objects based on RMI, we must instrument the source code of objects. When we design the tracing facility for RMI, we considered three requirements. First, low intrusive overhead: This is very important due to the fact that instrumentation with tracing facility changes the program behaviour. As a result this may lead to the so called "probe effect", or "Heisenberg effect"[8]. Second, transparent annotation: When we instrument the traced source code, an annotation code does not explicitly presented in it. Third, dynamically replacement: Out up-coming visualization system will use two phase hybrid post-mortem/on-the-fly technique. In this system, the target tracing focus can be changed under any circumstances. So, a tracing component can be replaced with another one. This enables the two phase post-mortem/on-the-fly technique. This can distribute the tracing overhead into two parts. Our Plug-in Sensor Model(PSM) adopts this three requirements. In figure 2, the term *sensor* is a small program for tracing a working program. Under any circumstances the sensor can be dynamically exchanged another one. In the post-mortem phase, a sensor will store tracing data into the local storage. In the on-the-fly phase, a sensor will transfer tracing data to the central visualizer. These sensors can be controlled by the central visualizer.

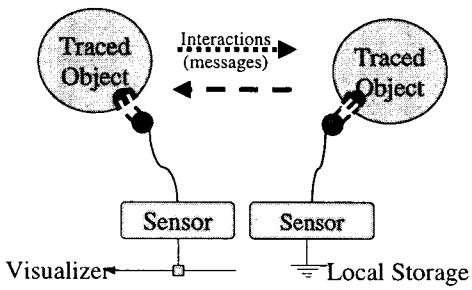

Figure 2: Plug-in Sensor Model

To contact with a sensor object, we added some codes into basic Java Object. Whenever some interactions are happened in the basic object, the annotation code accesses the sensor object. This implemented in the client stub and server skeleton code. It will be explained next section.

4 Implementation : TRMI

For this PSM, we insert instrument codes into client stub and server skeleton generated by `rmic` automatically. Our implementation is done by modifying nexusRMI included in Globus Project[9]. Traced data contain some useful information such as host name, object name, time-stamp and so forth. Present our implementation supplies remote object interaction information. When a Java object communicates with remote object, the Java serialization mechanism is used to offer the possibility of writing arbitrarily complex objects to a stream and reading them back. For object interaction ordering, we maintain logical clock. So, we piggybacked the logical clock onto the serialized object stream with transparency. This information will be used for global event ordering for correct visualization. Using this logical clock, we can know the dependency of objects. Figure 3 is the UML diagram of an example program.

`RMISolverImpl_Skel`
`RMISolverImpl_SkelUpcallThread`
`RMISolverImpl_Stub`
`RMISolverImpl_StubHandler` are the instrumented stub and skeleton classes. The `TimeStamp` class works as a sensor compoent. Whenever there are method invocations occur, the client stub and server skeleton works for it. Then, they have references of `TimeStamp` sensor object. Annotation of stub/skeleton class forwards some information to the `TimeStamp`. The `TimeStamp` stores the information into local storage or transfers central visualizer.

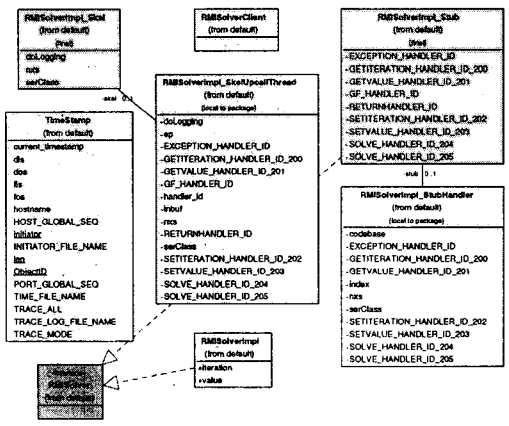

Figure 3: Example(UML notatioin)

5 Conclusion

We proposed Plug-in Sensor Model(PSM) and its implementation TRMI(Traced RMI). PSM supports dynamic tracing sensor under various circumstances. These are transparently equipped with Java object. To visualize overall remote interactions, we need some object event ordering method. In the future, we will add this capability.

References

[1] DANNY B. LANGE AND YUICHI NAKAMURA, *Object-Oriented Program Tracing and Visualization, IEEE* , pp. 63-70, 1997.

[2] EILEEN KRAEMER AND JOHN T.STASKO, *The 'Visualization of parallel systems: An overview., Journal of Parallel and Distributed Computing,* 18(2):105-117,June 1993.

[3] DOGULAS C.SCHMIDT, *The Distributed Visualization Environment .: A web-based telecom network management and visualization system using Java and Real-time CORBA, http://www.cs.wustl.edu/ schmidt/tri-dove.html*

[4] JERRY YAN, *Performance tuning with AIMS-an automated instrumentation, and monitoring system for multicomputers ,Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences,* January 1994.

[5] DANIEL A. REED, RUTH A. AYDT, ET AL., *An overview of the Pablo performance analysis environment. ,Technical report, Department of Computer Science, University of Illinos,* November, 1992.

[6] WEIMIG GU, GREG EISENHAUER, AND KARSTEN SCHWAN, *Falcon: On-Line minitoring and steering of large-scale parallel programs. ,Concurrency: Practive and Experience* , 1998.

[7] JIM FARLEY, *Java Distributed Computing ,O'reilly&Associates* , First Edition, January, 1998.

[8] PETER EADES, KANG ZHANG, *Software Visualization, World Scientific,* Series on Software Engineering and Knowledge Engineering Vol.7, 1996.

[9] GLOBUS PROJECT. http://www.globus.org