Monitoring and Controlling Distributed Applications with Relocatable Objects

Stephen Gilmore
The University of Edinburgh
stg@dcs.ed.ac.uk

Marco A. Palomino
University of Cambridge
map37@cl.cam.ac.uk

1. Introduction

The Java programming language and its environment are highly regarded as suitable choices for the development and deployment of distributed applications. Portability, an object oriented approach, support for multi-threading programming and a special set of libraries for remote procedure calls, are some of the features that make Java an excellent choice for the implementation of distributed applications. A considerable number of projects on Java distributed systems have already been completed. Here we can list Charlotte, SuperWeb, Javelin, Javelin++, Java/DSM, Voyager and Babylon.

Based on Java, the *Babylon* package [1] was designed to integrate into a single comprehensive system both new and previously researched ideas in Java distributed computing. It retains parts of the underlying mechanisms used in *Ajents* [2], a previous project from which Babylon took part of its source code. Babylon also has some innovative additions that are not currently supported by Java, such as *remote object creation*, *remote class loading*, *asynchronous remote method invocation* and *object migration*.

From the point of view of application designers, Babylon can be deployed as a layer of middleware which facilitates the use of clusters of computational resources allowing selective migration of code-containing objects from one computational environment to another.

From the point of view of programmers, Babylon can be seen as a collection of Java class libraries that provides the necessary software infrastructure to support straightforward access to information that is distributed, within organisations and all around the world.

It is relevant to note that all of the Babylon class libraries are written in pure Java. Among other things, this means that Babylon can be executed on any standard Java Virtual Machine (JVM), and it is potentially open to a vast audience of programmers. Besides, the Babylon package contains some useful features that are not currently supported by Java. Consequently, it reduces the difficulty of writing distributed programs.

Babylon is implemented using Java RMI. It has been shown, however, that Java RMI is very slow for high performance computing [4]. Most of the researchers in the field agree that it is inappropriate for interactive applications [3]. Given that the key features of Babylon are principally based on Java RMI, all the drawbacks and limitations of Java RMI are inherited by Babylon. The motivation behind this work is the submission of a contribution that eventually helps Babylon to become a better tool in the field of Java distributed object applications. To this end, we have developed graphical components to provide monitoring and management capabilities for a Babylon application. We have also made a number of technical extensions to the existing Babylon system, including upgrading it to the Java 2 security model, and we have adapted it to use a better implementation of RMI, because we believe that its dependency on Sun's RMI represents a drawback. Due to space limitations, we cannot state in this document all the details of our work, but readers may wish to take a look at [5] for a comprehensive description of our investigation.

2. Babylon monitoring interface

Babylon implements a scheduling system and allows the creation of a set of *Babylon servers* to control and govern the allocation of resources. The role of the Babylon servers is to offer a point of contact for the creation of objects on the machines where they reside. In the same way in which a World Wide Web browser is not able to access information stored on systems that are not executing an HTTP server, Babylon cannot offer its resources to computers that are not running a Babylon server.

The job of the *Babylon scheduler* consists primarily of supplying references to accessible Babylon servers that have identified themselves. In addition, it transparently manages the dynamically changing availability of the Babylon servers.

In the initial version of Babylon, the scheduler did not have any graphical user interface. In consequence, users had no convenient way to interact with the scheduler. In view of this, one of the extensions that we considered to improve the functionality of Babylon was the implementa-

Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)
0-7695-1582-7/02 $17.00 © 2002 IEEE

tion of a graphical user interface. This new element not only gives a better look to the whole application, but it also assists with administering and monitoring the operation of Babylon. Figure 1 displays a snapshot of the scheduler monitor, which is the first piece of our user interface.

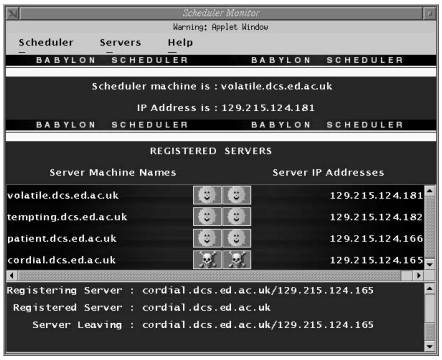

Figure 1. Scheduler monitor

Figure 2 shows an example of the server monitor, the second important piece of the user interface. The snapshot corresponds to a server that has just been started.

Figure 2. Server monitor

Keeping track of the objects residing at any time on the different Babylon servers is another capability of our user interface. We believe that it could eventually be useful to limit the creation of new objects and even to move existing objects when resources become heavily utilised.

3. Security framework

In previous versions of Java a security manager was responsible for determining which resource accesses were allowed to the different objects participating in an application. It was on the basis of that security model that Babylon was developed.

With the advent of Java 2, the Java security architecture became *policy-based*. This means that when code is loaded, it is assigned permissions in accordance with the security policy which is currently in effect. Each permission specifies a special kind of access to a particular resource, such as *read* or *write* permission to a file or directory, or connect access to a given host and port. The security policy that indicates which permissions are available for code can be initialised from an externally configurable policy file. No application can access a particular resource guarded by certain permission unless that permission has been explicitly granted in the security policy file.

It is mandatory to write a number of policy files to execute Babylon on the Java 2 platform. Permissions to accomplish some required network socket operations constitute the most important security concern for the Babylon scheduler, whereas runtime permissions represent the key security issue for the Babylon servers. Almost any Java object in Babylon can be used in a distributed context, but objects must be registered to take advantage of this functionality. This implies that programmers should first obtain a reference to the scheduler, and then consult it to access an available server where the objects can be created. Nevertheless, servers could not create objects if they did not have permission to read the files where the byte-code relevant to the object creation is stored. Therefore, we need to grant a `FilePermission` in the server security policy file.

4. Summary

The leading goal of this project was to enhance the functionality of Babylon to control the execution of object-based distributed applications. We developed graphical components to provide monitoring and management capabilities for a Babylon application. We also made a number of technical extensions to the existing Babylon system.

References

[1] M. Izatt. Babylon: A Java-based Distributed Object Environment. Master's thesis, York University, Canada, 2000.

[2] M. Izatt, P. Chan, and T. Brecht. Ajents: Towards an environment for parallel, distributed and mobile Java applications. *Concurrency: Practice and Experience*, 12(8):667–685, 2000.

[3] V. Krishnaswamy, D. Walther, S. Bhola, E. Bommaiah, G. Riley, B. Topol, and M. Ahamad. Efficient implementations of Java remote method invocation. In *4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'98)*, pages 203–241, O'Reilly & Associates, CA 95472, March 1998.

[4] C. Nester, M. Philippsen, and B. Haumacher. A more efficient RMI for Java. *Concurrency: Practice and Experience*, 12(7):495–518, 2000.

[5] M. Palomino. BabylonLite: An Improved Babylon Package. Master's thesis, The University of Edinburgh, Scotland, 2000.