A Framework for Testing Distributed Systems

Daniel Hughes
Computing Department
Lancaster University
Lancaster, UK.
+44 (0) 1524 594117
d.r.hughes@lancaster.ac.uk

Phil Greenwood
Computing Department
Lancaster University
Lancaster, UK.
+44 (0) 1524 592789
p.greenwood@lancaster.ac.uk

Geoff Coulson
Computing Department
Lancaster University
Lancaster, UK.
+44 (0) 1524 593054
geoff@comp.lancs.ac.uk

ABSTRACT

Thorough testing of distributed systems, particularly peer-to-peer systems can prove difficult due to the problems inherent in deploying, controlling and monitoring many nodes simultaneously. This problem will only increase as the scale of distributed systems continues to grow. This framework implements a test bed environment using a semi-centralized peer-to-peer network as a substrate for sharing resources made available from standard PCs. This framework automates the process of test-case deployment using a combination of Reflection and Aspect Oriented Programming. This allows 'point-and-click' publishing of software onto the test-bed. Our framework also provides a common monitoring, control and logging interface for all nodes running on the network. Together, these features greatly reduce deployment-time for real-world test scenarios. Automated insertion and removal of test code also ensures that the testing process does not compromise the correctness of the final system.

1. Introduction

This paper discusses a Java based testing environment that facilitates the testing of distributed applications with easy publication, monitoring and control of prototype systems on a peer-to-peer test-bed. Manual creation and maintenance of such tests is an extremely time consuming activity, especially where nodes are required to change their behavior dynamically. Such tests may require the creation of specialist control and monitoring tools. Furthermore, monitoring code must often be inserted throughout such prototype systems. The addition and removal of such code is time-consuming and error-prone. Our framework uses a combination of Reflection [1] and Aspect Oriented Programming [2] to automatically insert and remove code that is required for applications to interface with the testing framework's central monitoring and control interface.

2. Aspect Oriented Programming

Aspect-Oriented Programming is an emerging programming paradigm which extends Object-Oriented Programming (OOP). With OOP, some concerns cannot be cleanly captured in a single object and so become scattered across several objects. For example, an application monitoring concern such as the one discussed previously may require code to log specific method invocations or communicate them to a monitoring interface. It would be necessary for the said code to be scattered throughout the application, in each method the user wishes to monitor.

Such *crosscutting* concerns may be implemented in units of code known as *aspects*. Within the aspects, sections of code are created that are known as *advice,* these pieces of advice implement the concern. The advice is then *woven* into OO base-code at defined *joinpoints*. However, this still leaves the problem of identifying suitable join-points within the application where logging/monitoring code should be applied.

3. Reflection

We use Reflection to examine and extract information regarding the structure of the component being tested in order to facilitate the selection of joinpoints. Once the structure of the component is exposed to the user through a GUI, they can select the elements of the component they wish to monitor, and insert the appropriate monitoring aspect using the point-and-click interface. For example:

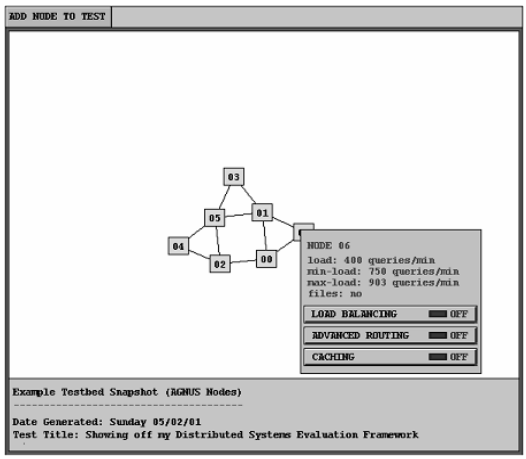
A peer-to-peer file-sharing system exists in which a developer desires to monitor the number of concurrent downloads in progress on each node. In this hypothetical system, a method '**`initiateDownload`**' is called when a new download begins and an integer variable '**D**' which represents the number of concurrent downloads is updated. Using framework tools on this component will expose a list of available methods. By selecting the **'`initiateDownload`'** method as the joinpoint and a variable-monitoring aspect for '**D**', a large-scale, real-world test can be published in just a few clicks so that each time a new download is initiated from any node it will be reported to the central monitoring/logging interface.

4. Peer-to-Peer Networking

A semi-centralised peer-to-peer network, using a single index server is used as the communications substrate for the framework. This is similar to Napster [3]. Nodes

participating in the test-bed network are typically standard general-purpose workstations. Each runs a small client application which allows it to host distributed applications.

The client registers with the index server that it is available to host a test process. When the developer publishes a process to the test-bed, they are informed by the server of the current number of nodes willing to host a new test. From this pool of nodes, the developer selects an appropriate test size, and 'publishes' the modified system to the network. The modified software is sent to each test-host using a simple peer-to-peer HTTP file transfer. When the software transfer is complete, the hosts execute the test-software and the inserted monitoring code will relay the status of each node on the network back to the server. The user can access this information in a dynamic graphical form via an associated Java applet or as text logs.



[Figure 1 – Graphical Monitoring Interface]

Following testing, framework tools are used to remove all communications code that was added for the test, ensuring that the correctness of the final application is not compromised by un-removed test code. The whole publication and testing process occurs without the developer having to access the source-code.

5. Implementation

Our Framework is currently implemented entirely in Java and only supports the testing of Java components. Java was chosen as the development language because of its inherent support for reflection and because several Java-based test candidates already exist in the department.

There are several Java AOP technologies, including AspectJ[4], JAC[5] and Hyper/J[6]. We chose to implement the system using AspectJ, as its joinpoint model closely follows our requirements and unlike JAC or Hyper/J, it does not introduce unnecessary overhead or constraints.

We chose to implement this system using a semi-centralised peer-to-peer model as it significantly reduced the complexity of the communication code that must be added to test processes.

6. Summary and Future Work

There have been a number of significant projects which have explored the creation of large-scale distributed test-beds [7], [8]; however, there has been very little progress on tool support for such large-scale testing. Furthermore, these existing test-bed implementations require specialist infrastructure, whereas our framework can be deployed over any existing network and makes use of the processing power available on regular workstations.

Our framework facilitates the deployment, monitoring and control of software on an extensible distributed test-bed. This is provided using a semi-centralized peer-to-peer network and the spare CPU-cycles of regular workstations. The automated insertion and removal of test code enables more rapid deployment of test scenarios and reduces the potential for test code to compromise the integrity of the final system.

There remain several open issues for the framework, these include: dealing with node-failure, protecting host nodes from the effects of malicious or poorly-written code, providing test code security and supporting programming languages other than Java.

5. References

[1] Coulson, G., "What is Reflection?", http://dsonline.computer.org/middleware/, 2003.
[2] Elrad, T. et al, "Discussing aspects of AOP", Communications of the ACM Vol. 44 No. 10 pp 33-38, 2001.

[3] Napster. www.napster.com.

[4] Kiczales, G. et al, "Getting Started with AspectJ", Communications of the ACM Vol. 44 No. 10 2001.

[5] Pawlak, R. et al, "JAC: A Flexible Solution for Aspect-Oriented Programming in Java", Reflection 2001, 2001.
[6] Ossher, H., Tarr, P., "Multi-Dimensional Separation of Concerns using Hyperspaces", IBM Research Report, 1999.

[7] L. Peterson, D. Culler, and T. Anderson. PlanetLab: A Testbed for Developing and Deploying Network Services, June 2002. Technical white paper, available at http://www.planetlab. org/pubs/vision.pdf.

[8] A. Avizienis et al, "The UCLA DEDIX System: A Distributed Testbed for Multiple-Version Software," Digest of 15th Annual International Symposium on Fault-Tolerant Computing, pp. 126-134, Ann Arbor, Michigan, June 1985