Testing Java Applets and Applications

White Paper

Mercury Interactive Corporation
1325 Borregas Avenue
Sunnyvale, CA 94089
408-822-5200
www.merc-int.com

Economics of Java Clients

There is a revolution in enterprise client/server computing—Java clients. Before Java, the cost of distributing and maintaining client software impeded universal access to enterprise computing assets. Information systems (IS) organizations bore the burden of installing, updating and supporting client software for each user. Only users whose job tasks directly and significantly benefited from access to a server-based asset warranted client software. Now, with Java clients, anyone with a Web browser can download an applet and access server-based assets. Distributing client software is simply a matter of posting the applet to a Web page. Updating the client software is just as easy. Much of the support costs are associated with assuring proper configuration of browser or stand-alone Java virtual machine, which are amortized across all Java client software that runs on top of these platforms.

With the near-zero distribution and support costs of Java clients, IS costs shift to client and server software quality assurance. The shift to Java poses three new problems for client quality assurance:

- **Cross platform support**. The advantage of Java is that anyone can download an applet and run it in any browser on any platform. Unfortunately, there are slight differences across platforms, virtual machine vendor implementations and windowing toolkits. IS has a new challenge in assuring that Java clients work properly for all combinations of these components.

- **Rapid enhancement**. With the instant distribution capabilities of Java, IS will be under pressure to rapidly deliver enhancements to client software. This requirement compounds the quality assurance problem of cross-platform support.

- **Customized clients**. Users in different departments will want to access some of the same systems. However, because they probably have different perspectives on the enterprise, each will want Java clients that cater to its perspective. IS will be under pressure to provide Java clients customized to the specific needs of each department. More versions of client software will further compound the quality assurance problem.

In addition to these client software quality issues, there are also two issues with server software quality:

- **Increased load**. The accessibility of clients will increase application load. Moreover, Java clients often imply n-tier architectures. Such architectures increase the number of components that can fail under the increased load. This issue will make load testing to identify interoperability, capacity and performance problems increasingly important.

- **Expanded use-cases**. With more users from many different departments accessing server-based assets, they will likely exercise application features in new and unexpected combinations. This issue will make comprehensive functional testing increasingly important.

The new client and server software quality issues make both functional and load testing crucial. IS organizations need a complete solution for functional testing of Java clients, load testing of Java-based applications and managing the increased volume of testing data. As a further complication, many applications will have a combination of traditional and Java clients for the foreseeable future. Therefore, not only does the testing solution have to work across Java hardware platforms, virtual machines and windowing toolkits, it also has to work with both Java and traditional clients. Only Mercury Interactive's Java testing solution can meet these rigorous requirements. Based on the proven WinRunner®, LoadRunner® and TestDirector® products, Mercury Interactive can provide both comprehensive functional testing and load testing across different Java configurations and traditional client platforms.

Java in the Enterprise

The first wave of Java deployment within the enterprise involves building Java clients for existing client/server systems. These existing systems fall into three major categories, as indicated in Figure 1:

- **Packaged Enterprise Resource Planning (ERP) applications**. Many enterprises have turned to packaged ERP applications from vendors like Baan, Oracle, PeopleSoft and SAP to give them enterprise-wide computing capabilities. The original versions of these applications used their own clients, or "fronts". These native fronts are generally very thin and can be maintained from a central location. Since Java is an open approach and offers similar advantages, vendors have begun to offer Java clients as well.

- **Custom two-tier applications**. Custom two-tier client/server applications use a custom client and a database server. The business logic is usually split between the clients and the database, yielding relatively fat clients. One of the drawbacks is the cost associated with distributing and maintaining fat clients. Many enterprises plan on moving to a three-tier architecture with thin Java clients.

- **Custom three-tier applications**. Custom three-tier client/server applications use a custom client, a custom application server and a database server. Usually, enterprises use a third-party application server development environment to create the clients and application server. Vendors of these environments have begun to offer Java clients as an option to achieve the flexibility of applet-based distribution.

In addition to converting these existing applications to Java clients, many enterprises are architecting new systems based on Internet and intranet browsers. These new applications assume HTML, Java and ActiveX clients plus business logic written in Java. A Java application server provides connectivity to the clients, executes business logic and provides connectivity to a wide variety of back-end resources.

In most applications, Java clients will not exist alone. ERP applications will still support their native fronts. Custom three-tier applications will still support their custom clients. During the transition to Java, client software may have parts that are implemented in a proprietary environment and parts that are implemented in Java. Pure Internet and intranet applications will have to support HTML clients and perhaps ActiveX clients as well. One can even imagine cases where developers will need to implement client

software that uses a combination of HTML, Java and ActiveX. The mix of clients will provide users a rich experience and allow developers to reuse client components implemented in a number of languages. However, assuring the quality of these clients across platforms, virtual machine implementations and windowing toolkits will be a significant challenge.

Beyond the client quality assurance challenge, there is also a system scalability challenge. Adding Java applet clients to the existing client base may increase system loads beyond known limits. Unfortunately, the most mission-critical enterprise systems will probably experience the greatest load increase because these systems contain the information that most people want. IS needs to gather comprehensive performance statistics before the deployment of Java clients to assess the supportable load and allow time for the deployment of additional hardware. Moreover, building Java clients for existing client/server systems add a new level of system complexity. ERP packages and custom two-tier applications will have a whole new tier. Pure Internet and intranet applications may implement a completely distributed architecture. Each new component in these systems adds another element of performance risk. IS needs to load test these new architectures to manage this risk.

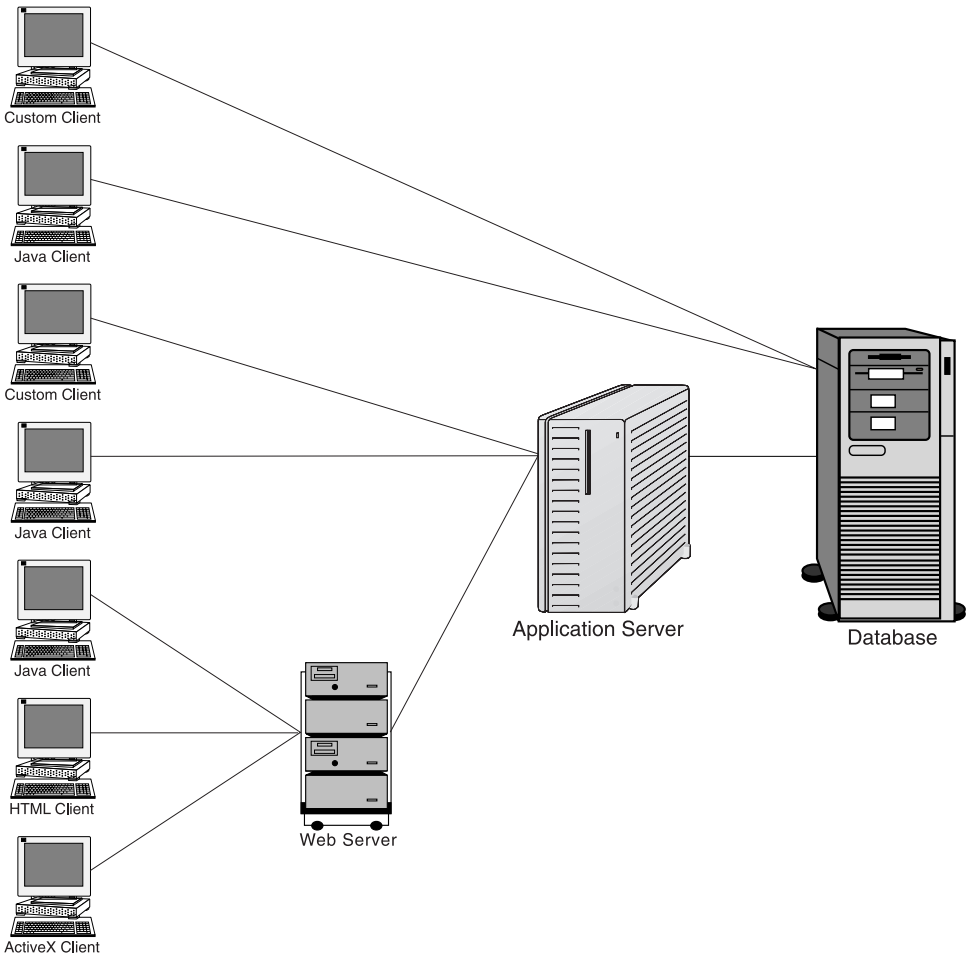

Fig. 1. Application with Java Clients

Java Testing Issues

Clearly, the introduction of Java is causing major changes in the client software distribution process and the complexity of application architectures. These changes indicate to three important testing issues: (1) rapid client deployment support, (2) cross-platform support and (3) complex architecture support.

One of the primary reasons for moving to Java clients is achieving time-to-market advantage. With downloadable clients, IS can respond to business change more rapidly because it can simply post new or updated client software to a Web page. The process for testing Java clients has to be equally rapid or the time-to-market advantage is lost. All client/server testing requires scripts as input. Scripts capture the interaction between the user and the client software. Typically, these scripts are created by recording actual users performing their job tasks using the application under test. Such scripts are a close approximation of the actual demands on the application. Test engineers usually supplement these scripts with ones designed to test specific parts of the system. Between these two types of scripts, test engineers can achieve complete coverage of application functionality.

To support rapid deployment of client software, testing tools have to minimize the time it takes to generate test scripts. The process of recording a script should be straightforward and quick. Scripts should work in the face of purely cosmetic changes. One can imagine that changing the position of a button could cause an inflexible tool to halt, forcing test engineers to re-record the script and re-run the test. A more flexible tool would simply note the button position change and continue with the script. Scripts should be easy to edit by test engineers so that they don't have to re-record a script to make small changes in a test. They should be able to store scripts in a repository and reuse the same script in as many different situations as possible.

Making the test script generation and management process as efficient as possible minimizes the time test engineers spend preparing tests. Testing tools also need to minimize the amount of time spent analyzing tests. One of the biggest time sinks in test analysis is filtering unimportant errors flagged by the testing tool. As discussed above, a testing tool should identify purely cosmetic changes as such so that test engineers don't spend time tracking down this type of problem. Another example of an unimportant error is change in date formats. Suppose a new version of client software presents date with a four-digit instead of two-digit year format, to address year 2000 issues. The testing tool should categorize the detected result as a format change, not an error. Beyond effective filtering of detected changes, testing tools need to support effective analysis across tests. Test engineers should be able to query a central repository and get information about the same test at different times or specific subsets of tests. This analysis capability makes it easier to pinpoint the source of errors.

The cross-platform combinations possible with Java can greatly add to test preparation and analysis time. If an enterprise uses Solaris, HP/UX, IBM AIX, Windows NT, Windows 95 or Windows 3.11, there are six hardware platforms. On the Windows NT and Windows 95 platforms, there are both Internet Explorer and Navigator browsers, as well as Sun's AppletViewer. There are also a host of different windowing toolkits, including Sun's JFC (Swing), Symantec's Visual Café, Oracle's Developer/2000, Sun's AWT and others. It would be arduous to require the recording of the same script on the

dozen or more combinations of these components. A single script should work with any mixture of these components. Moreover, if native ERP application fronts and custom client/server clients have the same logical layout as the Java clients, the same script should work for them as well. Another potentially arduous problem is having to manually execute tests on each different platform and move results to a central location. The testing tool should be able to centrally execute tests on any platform and automatically collate results in a central repository.

In addition to testing the different client combinations as efficiently as possible, testing tools need to facilitate the testing of the complex architectures discussed above. Some of these architectures have four or more tiers. Software flaws, hardware outages or network problems in any tier can slow the response of the system. Some Java application server products allow dynamic configuration of software and hardware resources, so testing tools cannot rely on rigid configurations. These complexities make load testing even more important than for traditional client/server systems. Testing tools need to measure end-to-end response times for the entire system. Moreover, test engineers need the ability to easily specify the measurement of different events so they can isolate the performance of different system components. In load testing, centralized results become extremely important for tracking improvements from performance optimization and detecting new of bottlenecks.

The complexity of Java client configuration along with the complexity of emerging application architectures make effective functional and load testing very important. Completing these tests fast enough to preserve the time-to-market advantage of Java client distribution is crucial.

Mercury Interactive Solution

Mercury Interactive's Java testing suite is the only solution that provides comprehensive functional testing, load testing and test management while providing features that minimize the test preparation and analysis for Java applications. Mercury Interactive uses the proven technologies from their traditional client/server test suite: WinRunner for functional testing, LoadRunner for load testing and TestDirector for test management. Moving to Java clients introduces an element of risk. Using unproven test tools would compound this risk. With installations at hundreds of customers, WinRunner, LoadRunner and TestDirector supply the proven foundation required from testing tools. In addition to the proven capabilities of these traditional client/server tools, Mercury Interactive provides an integrated functional testing, load testing and test management solution for Java environments. The same recorded scripts work for both functional and load testing. A centralized management console can schedule both functional and load tests then collect the results in a centralized repository. This total integration greatly reduces the time necessary to conduct a complete testing program.

One of the advantages of Mercury Interactive's Java solution is that it approaches testing from a business process perspective. The reason enterprises are turning to Java is to better support their business processes. It makes sense for the testing tool to support this perspective. This same process-oriented testing methodology is behind all of Mercury Interactive tools. This process has six steps. Each step has functional testing, load testing and test management tasks.

- **Gather requirements**. This step answers the question, "How do people use the system in real life?" Test engineers need to determine the response time requirements, identify user types and estimate usage patterns. They need to further break down the individual business processes each user type will perform with the software. This information lays the groundwork for recording scripts used in functional and load testing. Test management functions allow test engineers to input information directly from this step to generate a test plan.

- **Capture business processes**. For each business process, test engineers need to use a representative instance of that process. WinRunner and LoadRunner will record these transactions. This recording is used to generate both functional and load testing scripts.

- **Generalize business processes**. After capturing instances of business processes, test engineers need to transform them into scripts that emulate the behavior of multiple application users instead of just one. For an order entry business process, test engineers want to make sure that the test script will enter different customer information, product selections and quantities for each user that the tool emulates. After completing this phase, test engineers have functional and load testing scripts that will work on different hardware platforms, Java virtual machines and Java windowing toolkits. These scripts will also work with native fronts to ERP applications or custom clients for client/server applications. The scripts are stored in a central repository which tracks testing activity and results.

- **Build tests**. From the requirements gathering phase, test engineers will have functional and load testing plans that describe the set of tests they will run. For functional testing, test engineers will assemble a set of tests that perform all the business processes supported by the application plus special tests designed to exercise specific portions of the application individually. They will further specify the combination of hardware platforms, Java virtual machines, windowing toolkits and non-Java clients to be tested by each script. This approach guarantees complete coverage of application functionality on all possible platforms. For load testing, test engineers will assemble a set of tests that correspond to the different usage patterns identified in the requirements gathering stage. For instance, one usage pattern might have 50 users doing order entry, 25 doing product configuration and 100 checking order status. All the functional and load tests are stored in the testing repository

- **Execute tests**. With TestDirector's test management functions, test engineers can schedule functional and load tests to execute on all machines in the test cluster. They can even schedule them to run during down times such as midnight to 4 a.m. The testing repository gathers all results.

- **Analyze results**. With the data gathered during test execution, test engineers can get a summary of the different functional tests, including errors in screen layout, screen presentation and application responses. Summary of load tests allows them to chart response times to different user input, see scalability under load increase and detect server crashes. The testing repository allows test engineers to drill down into results, down to a complete replay of user input and system response. If they detect a problem, they have a complete history of all test results to help them identify the version of the software that introduced the problem and the tests affected.

The ability to use tests across different Java and non-Java clients decreases test preparation time even further. Mercury Interactive's solution builds a logical object map of the different widgets on screen, rather than recording cursor positions and keyboard strokes. It also understands the logical relationship between these different widgets. This understanding comes from the fact that the user interface is accessed at the API level. Mercury Interactive has layered this abstract object mapping technology on top of the Windows API, Motif and the various Java windowing toolkits. Therefore, an object map from an interface running on the Windows API will apply to an interface built on top of the AWT as long as the types of widgets and their logical relationships remain the same. No other tool works across the range of different Java and non-Java clients. These other tools require different scripts for different clients or won't work for some clients altogether.

Conclusion

Java clients offer enormous software distribution and maintenance advantages. However, they also present new quality assurance challenges. Each application will have more types of users. Each application will have more total users. Java clients will run on different hardware platforms, running different browsers, with different windowing toolkits. WinRunner provides the functional testing capabilities necessary to meet these challenges. Moreover, the applications themselves will be more complex. They will have more tiers. They will access more data sources. They will execute business logic packages into components. LoadRunner provides the load testing capabilities necessary to ensure such a complex system of software components can handle the higher user loads. Lastly, TestDirector provides the test management capabilities necessary to smoothly support the faster pace of Java software development. Only Mercury Interactive provides a complete solution for managing the risks posed by Java clients through an integrated functional testing, load testing and test management solution.

About Mercury Interactive

Mercury Interactive is the world's leading provider of enterprise application testing solutions. The company offers a comprehensive line of automated tools that address the full range of quality needs for testing client/server, e-business, Year 2000, Euro and ERP applications. Its testing solutions enable corporations, system integrators and independent software vendors to identify software errors more quickly and efficiently than traditional methods allow, enabling them to deploy software with confidence that it work effectively.

Mercury Interactive was founded in 1989 and went public in 1993. The company is based in Sunnyvale, California, with additional offices providing research and development, sales, and support around the world. The Company's common stock trades on the NASDAQ National Market tier of The Nasdaq Stock Market under the symbol MERQ.

For more information on Mercury Interactive products and services, please contact your local Mercury Interactive sales office or call **1-800-TEST911**. You can also visit our Web site at **www.merc-int.com**.

384-BR-JAVAWP1