Software Test Automation for Multi-Platform Client/Server Applications

Huey-Der Chu , Feng-hao Liu* ,C.C.Yang*

Graduate School of Defense Informati
National Defense Management College
ChungHo, Taipei 23500, TAIWAN
jchu@mis.ndmc.edu.tw

*Software Engineering Laboratory
Electronic Department, National Taiwan
University of Science and Technolog

Abstract

To assist a solution to the problem of the test environment spanning multiple platforms, this paper applies mobile agents to automated test execu tion. A test driver can be launched by a mobile agent to remote client sites to run the tests. During the testing, the mobile agent will use the Network Class Server to dynamically load th classes relevant to this test. The test result on each client site will be sent back by the mobile agent. The mobile agent roams across different platforms and finally it arrives at th application server site to bring back the trace file for inspecting the interaction behaviour among clients. This work can really let th e tests run on multiple client sites across different platforms.

Keywords: *Mobile Agents, Automated Test Execution, Integrated Test Environment*

1 Introduction

Current testing tools with capture/playback paradigm have some limitations for client/server applications [10,12]. The common one is a lack of consideration of the real world operating environment across multiple platforms. These testing tools emulate a multi-user environment and ends at the client site but are not designed to test the server, therefore, their products may not provide a way to test the effect of multiple users of the software. Moreover, these testing tools focus on two-tier client/server applications. However, the Gartner Group found 80% were planning for multi-tier (at least three-tier) client/server applications [12]. For client/server applications, some test scripts for middleware can be very hard to capture/playback automatically [9], for example, the communication mechanism between clients and servers uses technology like an RPC pr otocol that current capture/playback tools cannot effectively capture.

This paper introduces a flexible infrastructure for mobile agent computing: VISITOR [2], which can support flexible communication and co-operation between mobile agents and local agents which may provide some services through the agent broker. Furthermore, combining with the Java Remote Method Invocation (RMI), mobile agents can make use of distributed objects to accomplish such tasks as sending the results back to the home machine. Th ese local agents are like offices which receive visitors and provide some services to them, while the mobile agents are like visitors which move around one office to another for their particular goals.

Based on VISITOR, a simple three-tier bankin application with an integrated test environment has been implemented, which is big enough to address middleware testing issues such as Java RMI and JDBC. The test driver is launched by a mobile agent to remote client sites to run the tests. During the testing, the mobile agent will use the Network Class Server to dynamically load the classes relevant to this test such as the Test Data Generator and Test Results Validator. The test result on each client site will be sent back by the mobile agent. The mobile agen t roams across different platforms and finally it arrives at the application server site to bring back the trace file for inspecting the interaction behavior among clients. The application of VISITOR to automated test execution can let the tests really run on multiple client sites across different platforms.

Firstly, in this paper, the problem of current testing tools is addressed.; secondly, the concept of mobile agents is introduced; thirdly, the framework of VISITOR is proposed fourthly, the application of VISITOR to the automated test execution is described; fifthly, the concept of automated test execution through mobile agents across multiple platforms is illustrated on a three-tier client/server application with Java RMI and JDBC and finally, summarizes the work and offers suggestions for further study.

2 Current Testing Tools

Test execution is a process of feeding test input data to the application and collecting information to determine the correctness of the test run. It is natural to assume that automating test execution must involve the use of a test execution tool which requires an environment to run it, t accept inputs and to produce outputs [7]. Some tools require additional special-purpose hardware as part of their environment; some require the presence of a software development language environment.

For a sequential software, this process can be accomplished without difficulty. Many testers do not have strong programming skills. This combined with the repetitive nature of much testing, l eads people to use

capture/playback techniques and tools. All capture/playback tools are an elaboration and more modern implementation of these simple ideas. Although simplistic, the importance and impact of capture/playback should not be under-estimated. These tools are often the first test tool a tester may see and are also the primary means by which the test process is migrated from a purely manual process to a mostly automated process.

Indeed there are many tools that allow test scripts to be recorded and then played back, using screen captures for verification. However, there are some inherent problems with the capture/playback paradigm [11,13]: Firstly, test automation is only applied at the final stage of testing when it is most expensive to go back and correct the problem. Secondly, the testers do not get an opportunity to create test scripts until the application is finished and turned over and thirdly, the problem that always crops up is that application modifications are made, invalidating the sc reen captures and then the interface controls change, making playback fail. Moreover, for client/server applications, some test scripts can be very hard to capture/playback automatically [12]: Firstly, the communication mechanism between clients and servers uses technology like an RPC protocol that current capture/playback tools cannot effectively capture. Secondly, simulation and thirdly, there are non-deterministic behaviours in a distributed application. Repeated executions of a distributed application w ith the same test script may execute different paths and produce different results. This is called the non -reproducible problem. Therefore, some mechanisms are required in order to exercise these test scripts.

3 Mobile Agents

An agent is an object that is autonomous enough to act independently even when the user or application that created it is not available to provide guidance and handle error. Agents can receive requests from external sources, such as other agents, but each individual agent decides whether or not to comply with external requests. In the computer world, an agent is a computer program whose purpose is to help a user perform some tasks (or set of tasks) [8]. To achieve this aim, it maintains a persistent state and can communicate with its owner, other agents and the environment in general. Agents can do routine work for users or assist them with complicated tasks. In addition, they can mediate between incompatible programs and thus generate new, modular and problem-oriented solutions thus saving work.

Mobile agents [1,3] provide a new alternative paradigm for distributed object computing on the WWW. A mobile agent is a computer object that can move through a computer network under its own control, migrating from host to host and interacting with other agents and resources in order to satisfy requests made by its clients. They may move around on behalf of their users seeking out, filterin and forwarding information or even doing business in their own name. Possible applications for mobile a gents include information retrieval, data -mining, network management, electronic commerce, mobile computing, remote control and monitor, etc. Therefore, mobile agents show a way to think about solving software problems in a networked environment that fits more naturally with the real world.

The concept of mobile agents is in contrast to the concept of Java Applets. In the latter case, a program is downloaded from remote computers to execute locally, while in the former, a program is sent to remote machines to execute remotely, When mobile agents execute remotely, there may not be any transactions in the home machine. The advantages of mobile agents are [3]: firstly, they offer an effective paradigm for distributed applications, particularly in partially connected computing; secondly, they can provide a pervasive, open, generalized framework for the development and personalization of network services; thirdly, they move the programmer away from the rigid client/server model to the more flexible peer -to-peer model in which programs communicate as peers and act as either clients or servers depending on their current needs and fourthly, they allow ad-hoc, on-the-fly applications that represent what would be an unreasonable investment of time if a code had to be installed on each network site rather than dynamically dispatched.

Nowadays, there are already some frameworks for mobile agents, such as the Aglet and the Java -to-go. They all support dispatching a segment of code to remote machines to execute, however they do not give proper support to the co -operation between mobile agents and services in remote machines. This next section introduces a flexible infrastructure for mobile agent computing: VISITOR [2], which can support flexible communication and co-operation between mobile agents and local agents which may provide some services through the agent broker. Furthermore, combining with the Java Remote Method Invocation (RMI), mobile agents can make use of distributed objects to accomplish such tasks as sendin g the results back to the home machine. VISITOR shows a paradigm for service-providers to provide services and for service-clients to get services in a networked environment that fits more naturally with the real world.

The application of VISITOR to soft ware testing, Mobile Testing Agent [5], has been implemented and can be downloaded at the MObile Software Testing (MOST wesite (http://www.casq.org/most/) constructed and maintained by Hue -Der Chu 1998.

4 The Architecture of VISIT

The architecture of the VISITOR is shown as in Figure 1 which consists of a network of agent servers, agent clients and a security server.

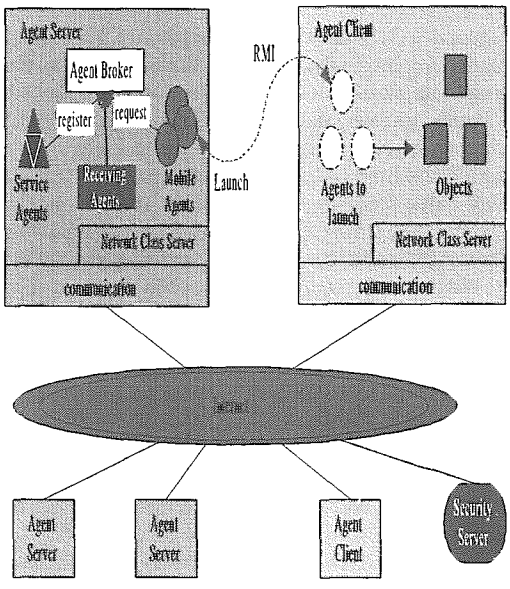

Figure 1: The architecture of VISITOR

These components communicate with one another based on Java sockets. Agent servers are destinations which mobile agents want to visit. Agent Servers are also the hosts which accommodate mobile agents and provide services to them. Agent Clients are applications which launch mobile agents to the agent servers for accomplishin their particular tasks. In this framework, the agent servers are like offices which receive visitors and provide some services to them, while the mobile agents are like visitors which move around one office to another for their particular goals.

4.1 Agent servers

In each agent-server, there are five types of components: The Agent Broker (AB), service agents, the receiving agent, mobile agents and the network class server.

An AB is a stockbroker among agents. All other agents have to be registered with the AB. The AB keeps them as resources. When an agent is created, it sends a message to the AB to register its existence and address. When an agent A wants to communicate with another agent B, A first transmits a message to the AB to ask B's address. The AB would acknowledge with B's address if B exists. When B first receives A's message, it also need to ask the AB for A's address. Afterwards, A and B would communicate with each other directly.

Furthermore, if an agent, for example a service agent, could provide some service, it would send the AB a message to register that service. When some agent, for example a new coming mobile agent wants the service, it would request the AB. If the service has been registered, the AB would return the agent's address that can provide that service. Then, they would dialogue directly as normal.

Service agents provide services for other agents. When they are created, they would register the service with the AB which they can provide. The services they can provide are various, from general information services to particular commercial services.

It is the receiving agent that is responsible for receivin and instantiating coming mobile agents. It also creates execution environments and forks a thread for the agent run. There is only one receiving agent in each agent-server. For the structure of the receiving agent, see section 5. 1.

Mobile agents come from remote agent clients. When they arrive, the receiving agent creates the execution environment for them and they would register with the AB. Together with main classes, a knowledge bas e which include initial information is sent. The receiving agent will save this knowledge as a specific file. A mobile agent will run in a separate thread to accomplish its tasks. It can als make use of services which are provided by execution environments or service agents. For the structure of mobile agents, see section 5.2.

The Network Class Server listens to the network. If there is a request for loading a class from this machine, it is responsible for finding, loading and sending the class to the destination. When a mobile agent is launched, only the main class is sent. the auxiliary classes are loaded on demand from the home machine or the previous machine, where a network class server is set up.

4.2 Agent clients

Agent clients design and launch mo bile agents for accomplishing their particular tasks. The clients may be located in an agent-server or in a separate machine. For the latter, a network class server has to be set up for remote class loading. In the case where there is no network class server set up, the agent launcher has to send all class of the agent, or the class loader would fail.

Arriving at remote agent servers, mobile agents can execute home transactions by the Java RMI. For example, when a mobile agent retrieves information in rem ote agent servers, it can make use of the RMI to display the result on the home machine simultaneously.

The picture above characterises a flexible agent-oriented method of constructing client applications, producing a ne paradigm for distributing comput ing.

4.3 Security server

The security Server is listening to the network. When clients want to launch a mobile agent for accomplishin their particular tasks, they have to register with the securit server to gain a key, which the mobile agent will bring ith it. The agent servers will check the key to see whether or not it is valid. If the key is valid, the process will continue, if not the server will send back an error message to the client.

5 A General Structure of Agents

The static structure of an agent is designed following the Java Agent Template (JAT). An agent consists of three parts: a message handler, a resource manager and a knowledge base. The message handler sends and receives Knowledge Query and Manipulation Language (KQML) messages by the communication interface *Comminterface*. The message handler is also responsible for message processing.

The resource manager is responsible for managing resources which the agent possesses. There are five types of resources: Languages, interpreters, classes, files and addresses in the JAT.

The knowledge base includes the initial information of agents and the information about the services which it can provide. When the agent moves from one machine t another, the information in the knowledge base will move along.

An agent executes within a *AgentContext* which is the execution environment of the agent. Agents could make use of services in the agent-server by the *ContextInterface* which is implemented by the *AgentContext*. When an agent arrives at a new agent-server, the receiving agent will initiate the agent with the knowledge base, which is sent with the underlying agent. When an agent leaves the machine, it will clean up the environment. The *initiate* and *cleanup* methods are provided by the *AgentInterface*. Dynamically, an agent is a thread. When an agent moves t a new agent-server, a new thread is created, on which the agent is running.

5.1 Structure of the Receiving Agents

The receiving agent inherits from a general agent but the receiving agent has its spec ific functions in VISITOR. The layers of a receiving agent are as shown in Figure 2.

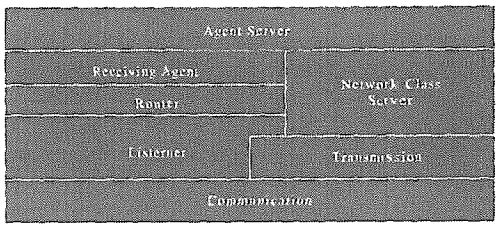

Figure 2: Layers of a Receiving Agent

When the *receiving agent* starts up, it forks a thread to execute the *Router*, which in turn forks a thread to execute the *Listener*. Based on Java Sockets and severSockets , the *Transmission* layer provides semantics of the agent -packet transmission. The *Listener* is monitoring the network to see if a new packet is coming. If so, the *Listener* makes use of the methods provided by the *Transmission* layer to receive the packet and pass it to the *Router*. The *Router* unpacks the packet and instantiates the coming agent first, then checks if the underlying machine is the destination of the agent. If not, the *Router* would rout the agent to the correct machine. If it is true, the *Router* would initiate the agent, create its execution environment and pass it to the *receiving agent*. The *receiving agent* forks a new thread to execute the new coming agent.

It is the Network Class Server (NCS) that implements the dynamic class loading. The pri nciple of dynamic class loading is shown as in Figure 3.

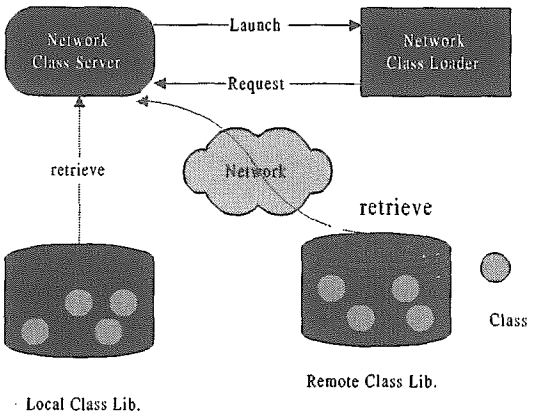

Figure 3: Dynamic Class Loading

The NCS is listening on the network, when a Network Class Loader (NCL) asks for classes it will find, load and transport the classes. The NCS not only can load classes from the local class library but can also load classes from a remote class library.

The layer structure of a NCS is as shown in Figure 4. Like the agent-server, it is also based on Java *Sockets* and *serverSockets*.

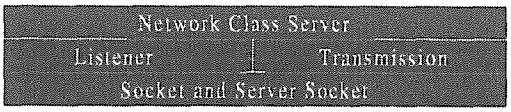

Figure 4: The Layers of Network Class Server

In the context of VISITOR, when the *Router* in the agent server *instantiates* a coming mobile agent, it will load the classes relevant to the agent dynamically.

5.2 Communication between Agents

Agents communicate with each other using the KQML [9], which is a high-level language intended for the run -time exchange of knowledge between intelligent systems.

Logically the KQML message consists of three layers: the content layer, the message layer, and the communication layer. The content layer includes the actual content of the message in the programs' own knowledge representation of the message. KQML can carr expressions encoded in any representation language such as the Knowledge Interchange Format (KIF), the KQML or even ASCII strings.

The communication layer encodes a set of message features which describe the lower lev el communication parameters, such as the identity of the sender and recipient, and a unique identity associated with the communication.

It is the message layer that is used to encode a message that one application would like to transmit to another. The message layer forms the core of the KQML and determines the kinds of interaction one can have with a KQM -speaking agent. A primary function of the message is to identify the protocol to deliver the message and t supply a speech act or *performative* which the sender attaches to the content (such as an assertion, a query, a command or any of a set of known *performatives*). In addition, since the content may be opaque to the KQM -peaking agent, this layer also includes optical features which describe the content language, the ontology it assumes and some type of description of the content such as a descriptor naming a topic with the ontology.

Syntactically, a KQML message is a ASCII string called a *performative*, which consists of a *performative's* name and a list of its parameters. A parameter is represented as keyword/value pair. The keyword, that is the parameter name must begin with a colon and must precede the corresponding parameter value.

Here is an example of a KQML message, which is used as an initial message in our framework

```
(evaluate :sender kbase :receiver agent
          :language KQML :ontology agent
          :content( tell-resource :type address
                    :name AB
                    :value mis.ndmc.edu.tw:5001))
```

In this message, the KQM *performative* is the *evaluate*, the content is (tell-resource :type address :name AB :value mis.ndmc.edu.tw:5001), another KQML message which tells the agent that the AB's address is *mis.ndmc.edu.tw:5001*, the ontology assumed is *agent*, the receiver and sender of the message are agent and *kbase* respectively, and the content is written in the language KQML.

The value of the content keyword is content level, the values of :sender and :receiver belong to communication level, and the *performative's* name (evaluate) with :language and :ontology form message layer.

When an agent ClientA moves to an agent-server, it would transmit a message like the following to the AB for telling its existence

```
(evaluate :sender Client
          content (tell-resource
                    :type address
                    :name Client
                    :value mis.ndmc.edu.tw:54100)
          ontology agent :receiver AB
                    :language KQML)
```

Suppose that there was already another a gent ClientB which sent the following message to the AB when it started up.

```
(evaluate :sender ClientB
          :content (tell-resource
                    :type address
                    :name ClientB
                    :value mis.ndmc.edu.tw: 4103)
          :ontology agent :receiver AB
                    :language KQML)
```

When the agent ClientA wants to communicate with the ClientB, it would first send the following message to AB

```
(evaluate :sender Client
          :content (ask-resource
                       :type address :name ClientB)
          :ontology agent :receiver AB
                       :language KQML)
```

The AB would answer with the message below

```
(evaluate :sender AB
          :content (tell-resource :type address
                   :value mis.ndmc.edu.tw:
                   :name ClientB)
          :ontology agent :receiver AB
                   :language KQML)
```

After that, the ClientA would dialogue with Client directly.

6 Automated Test Execution Through VISITOR

6.1 A Simple Banking Application

A banking application is an embedded software system which is commonly seen inside or outside banks to drive the machine hardware and to communicate with the bank's central banking database. This app lication accepts customers requests and produces cash, account information, database updates and so on. In this Section, a Simple Banking Application (SBA) will be designed as a 3 -tier client/server application.

Within a banking enterprise, more specifically a corporate and distributed database collection for the personal data of customers, the balance status of customers, the password data and account type data. The corporation seeks to assimilate their data sources into one virtual data store and access it through a common interface.

There are four business activities at this application: check balance, deposit money, withdraw money and print the statement. This standard transaction will accept customer requests (checking, depositing, withdrawing and printing) after the customer has input the account id, the account type and the correct password on the Client site. SBA will retrieve the balance from the database on the Database Server site, process the request on the Application Server site and save th e balance back to the database. It also will produce the balance or print a banking statement to the customer. It has been implemented with an integrated test environment using Java RMI and JDBC [4].

6.2 The Integrated Test Environment

Based on the framework for automating statistics-based testing [5], a Statistics-based Integrated Test Environment (SITE) is built and can provide automated support for the testing process, to address two main issues, deciding when to stop testing and determining how good the software is after testing. It consists of computational components, control components and an integrated database. The computational components will include the Modeller for modelling the applications as well as the quality plan, the SIAD/SOAD Tree Editor for specifying input and output messages, the Quality Analyst which includes the statistical analysis for determining the sample size for the statistical testing and the test coverage analysis for evaluating the test data adequacy, the Test Data Generator for generating test data, the Test Tracer for recording testing behaviours on the server side and the Test Results Validator for inspecting the test results as well as examining the "happened before" relationship. The architecture of SITE is as shown in Figure 5.

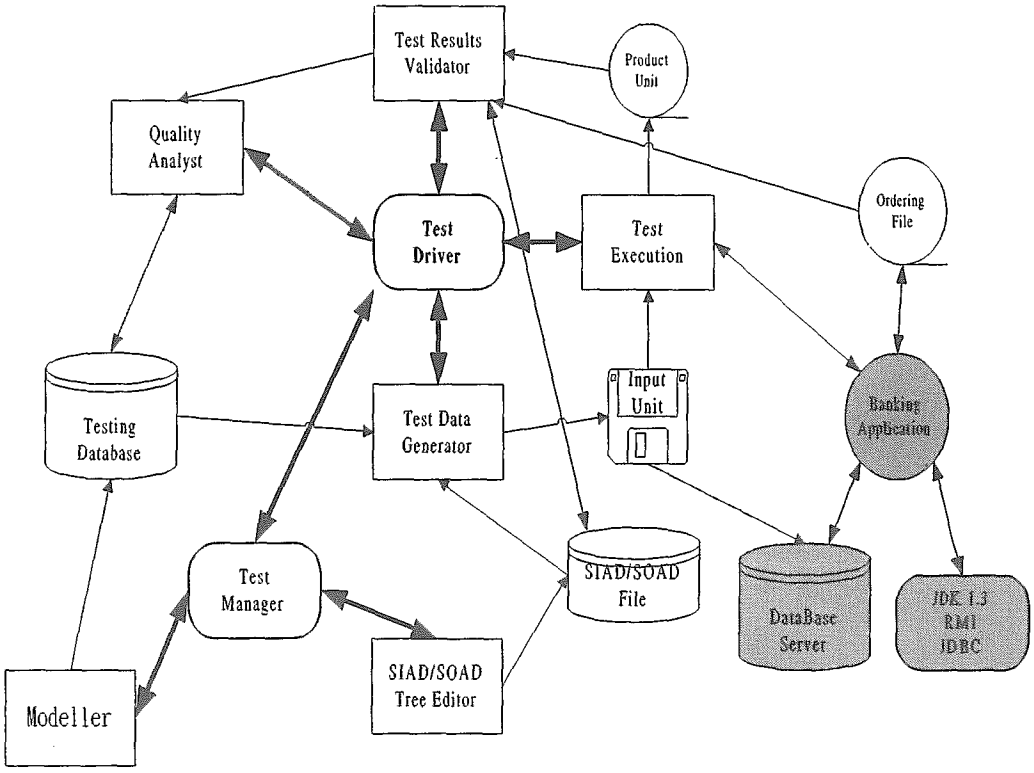

Figure 5: An Integrated Test Environment for the Banking Application

There are two control components, the Test Manager and the Test Driver. The Test Manager receives commands from the tester and corresponds with the functional module t execute the action and achieve the test requirements. It executes two main tasks: data management and control management. In data management, the Test Manager maintains an integrated database which consists of static data files and dynamic data files which are created, manipulated and accessed during the test process. The static files include a SIAD/SOAD tree file, a random number seed file and a quality requirement file. The dynamic files include an input unit file, a product unit file, a test ordering file, a defect rate file, a file for the defect rate range and a sample size file.

In control management, the Test Manager controls three main functional modules: the Modeller, the SIAD/SOAD Tree Editor and the Test Driver. The Modeller is used for receiving the test plan such as test requirements and test methods from the users, creating test plan documentation and saving some values for the testing database. The documentation produced by the Modeller provides support for test planning to the Test Driver as well as the SIAD/SOAD Tree Editor for specifying messages among events. The SIAD/SOAD Tree Editor is used to create the SIAD/SOAD tree file that can be used to describe the abstract syntax of the test cases as well as to trace data occurring during the test. The SIAD/SOAD tree file provides the structure to the Test Data Gener ator for generating input unit and the Quality Analyst to inspect the product unit. The Test Driver executes the main task of testing which includes the Test Data Generator, the Test Execution, the Test Results Validator and the Samplin Processor. It has been implemented with an integrated test environment using Java RMI and JDBC [4].

6.3 The Application with VISITOR

The application of VISITOR to the automated test execution on the banking application is as shown in Figure 6.

The test driver sets up the test execution environment for the banking application, initiating the Test Data Generator to generate an input unit, sending it to the Test Execution t execute the application and getting the product unit and delivering it to the Test Results Validator. The test driver is launched by a mobile agent to remote client sites to run the tests. During the testing, the mobile agent will use the network class server to dynamically load the classes relevant to this test such as the Test Data Generator and the Test Results Validator. The test result (pass/fail) on each client site will be sent back by the mobile agent with Java RMI. The mobile agent roams across different platforms and finally it arrives at the application server site to brin back the paths trace file for inspecting the testing order.

This framework has been implemented with the Java Agent Template (JAT) and the Java Remote Method Invocation (RMI). The JAT provides a fully functional template for constructing agents which communicate peer-to-peer with a community of other agents distributed over the Internet. However, JAT agents are not migratory but rather have a static existence on a single host. As an improvement, the Java RMI is used to let JAT agents dynamically migrate to foreign hosts in this implementation As a result of the Java RMI not currently working effectively well on the Netscape Browser currently, the implementation of MTA (Mobile Testing Agent), the name of a mobile agent for the automated testing in this implementation, is not available with Java Applet, but with stand-alone style. It can be downloaded at the MObile Software Testing (MOST) web site (http://www.casq.org/most/) which is under the web site for Chinese Association for Software Quality (CASQ) constructed and maintained by Hue -Der Chu 1998.

6.4 Discussion

Current testing tools with the capture/playback paradigm emulate a multi-user environment and ends at the client site but are not designed to test the server, as referred to in the Section 2. The application of mobile agents to automated

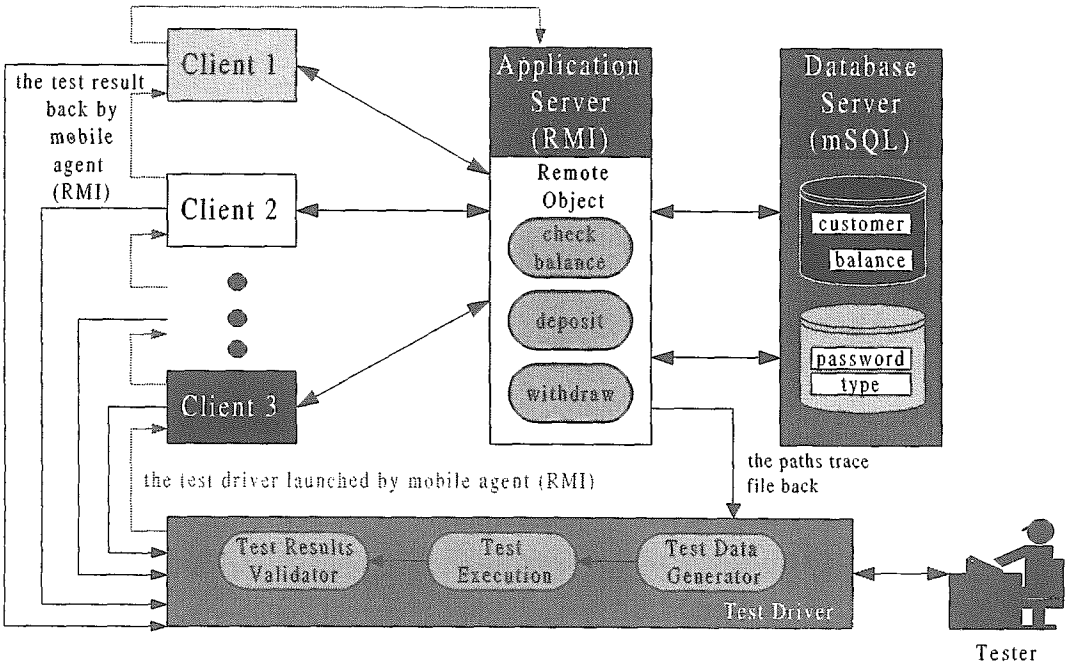

Figure 6: Automated Test Execution Through a Mobile Agent

test execution can let the tests really run on multiple client sites across different platforms and go to the server site t bring the paths trace file back to the home site for inspection of the test ordering.

Often, automated testing is introduced very late in the implementation process and is restricted to regression testing. The earlier the testers incorporate an automated test approach into the development process, the greater the return on the investment. In this implementatio n for automated testing through mobile agents, changin syntactic structures such as screen layouts does not interfere with test execution since any changes for these structures can be done by the SIAD/SOAD Tree Editor before the test execution. In other words, some testin activities involving components such as the Modeller and the SIAD/SOAD Tree Editor can be done early.

However, VISITOR cannot be used to solve the non-reproducible problem in this paper. Therefore, a complementary mechanism is required in order to achieve the global goal of client/server testing.

7. Conclusion

To assist a solution to the problem of the test environment spanning multiple platforms, the concept of mobile agents was introduced in this paper. A mobile agent is a computer object that can roam over the Internet under its own control migrating from host to host and interacting with other agents and resources in order to satisfy requests made by its clients. Based on the concept of mobile agents, the test driver can be launched by a mobile agent to remote client sites to run the tests and the paths trace file on the server side can also be sent back to the user for inspecting the test ordering. This concept has been implemented on a 3-tier banking client/server application with Java RMI and JDBC. It is completely different from current automated testin tools. The major advantages of this approach are the interaction behaviours between clients and server can be recorded in a paths tracer file which can be inspected and the tests can be really run on multiple clients across different platforms.

In practice, the software development methodologies typically employ a combination of server software testin methods, techniques and tools. The need to combine testin tools is further visible. Therefore, before the multi -user test executions, a dynamic test plan is needed to classify the testing types between multiple users on different platforms and will be extended in future work..

References

[1]Chen, J., "A flexible framework for mobile agent systems,". Available a http://www.casq.org/most/chen.ps.

[2]Chen, J., Greenwood, S. and Chu, H., "VISITOR: Java-based Infrastructure for Mobile Agent Computing," *Proc. in 10 th International Conference on Software Engineering and Knowledge Engineering (SEKE'98)* , June 18-20, 1998, San Francisco, USA.

[3]Chess, D., Harrison, C. and Kershenbaum, A., "Mobil agents: Are they a good idea?" *Lecture Notes in Computer Science 1222*, 25-45.

[4]Chu, H., *Distributed Testing: Towards Qualit Programming in the Automated Testing of Distributed Applications*, Europe Arts, Science & Culture Publishin C. Ltd., Feb. 1999, ISBN 1 -902409-07-8.

[5]Chu, H., Dibson, J.E., and Liu, I.C., "FAST: a framework for automating statistics-basd testing, " Software Quality Journal, 6(1), 13-36.

[6]Chu, H., Dobson, J.E., Chen, J. and Greenwood, S., "The Application of Mobile Agents to Softwar Testing," *Proc. in 15 th International Conference an Exposition on Testing Computer Software (TCS'98)* , June 8-12, 1998, Washington, D.C., USA.

[7]Fewster, M. and Graham, D *Software Test Automation: Effective Use of Test Execution Tools (ACM Press)* , Addison-Wesley Pub Co; Aug. 1999, ISBN: 0201331403.

[8]Lingnau, A. and Drobnik, O., "An HTTP -based infrastructure for mobile agents," Available a http://www.w3.org/pub/Conferences/WWW4/.

[9]Mayfield, J., Labrou, Y. and Finin, T., "Evaluation of KQML as an agent communication language," Availabl at http://www.cs.umbc.edu/lait/papers/kqml-eval.ps.

[10] Mooney, K. and Chadwick, D., "Overcoming the Challenges of Testing Client/Server Applications, Available a http://www.rational.com/support/techpapers/challenges/.

[11] Pettichord, B., "Success with Test Automation," *Proc. in 9th International Software Quality Week (QW'96)*, May 1996, San Francisco, USA.

[12] Quinn, S.R. and Sitaram, M., "Shrink-wrapped and custom tools ease the testing of client/serve applications," *Byte*, Sept. 1996, 97-102.

[13] Zallar, M., "Automated Software Testing - A Perspective," *Proc. in 10 th International Software Quality Week (QW'97)*, May 1997, San Francisco, USA.