ISSN: 2454-1907
DOI: https://doi.org/10.29121/ijetmr.v5.i4.2018.220

IJETMR
International Journal of Engineering Technologies and Management Research
A Knowledge Repository

AN INTELLIGENT FRAMEWORK FOR DYNAMIC TEST PLAN OF CLIENT/SERVER APPLICATIONS

Huey-Der Chu [*1]
[*1] Department of Information Technology, Takming University of Science and Technology, Taiwan Country

Abstract:
To assist a solution to the problem of the test environment spanning multiple platforms, this paper proposes a decision support framework with the blackboard model to integrate all complementary features into a single automated test environment for multi-platform client/server applications. Before testing client/server applications, the input into this framework are testing tools with different approaches and client sites which are going to run the test. The planning agent will make a decision dynamically and produce a testing plan to allocate testing tasks to these testing tools to client sites. Two complementary features for testing client/server applications are illustrated in this paper to demonstrate how the framework works. The concept of mobile agents is applied to launch the test driver to different client sites, execute the tests and bring back the test results from client sites as well as the trace file from the server site for inspecting the interaction behavior among clients. Based on the multicast framework, the same test data can be broadcasted to multiple clients sites to run the tests simultaneously and the test results can be returned from client sites for examining the problem of repeated executions.

Keywords: Blackboard Model; Automated Test Execution; Mobile Agents; Chat System.

Cite This Article: Huey-Der Chu. (2018). "AN INTELLIGENT FRAMEWORK FOR DYNAMIC TEST PLAN OF CLIENT/SERVER APPLICATIONS." *International Journal of Engineering Technologies and Management Research,* 5(5), 1-10.

1. Introduction

Client/Server applications are traditional applications re-cast for a new environment of multiple inter-linked computers and have been designed as systems whose data and processing capabilities reside on multiple platforms, each performing an assigned function within a known and controlled framework contained in the enterprise. Applications can now be broken into pieces that are common to more than one application and therefore stored, maintained and executed in a central location (a server), with the results sent back to the requesting client. The complexity of the client/server makes testing more difficult and poses new challenges to the development organization. Because each component can not always be tested as a single unit, integration testing becomes the lowest meaningful level testing. Defining all test conditions for a set of integrated functions is difficult enough with structured programs and procedures, let alone those that have been distributed across client and server platforms.

Software testing is characterized by the existence of many methods, techniques and tools, that must fit the test situation, including technical properties, goals and restrictions [9]. In practice, the software development methodologies typically employ a combination of several software testing methods, techniques and tools. There is no single ideal software testing technique for assessing software quality. We must ensure that the testing strategy is chosen by a combination of testing techniques and tools at the right time on the right world [3]. Therefore, we present a blackboard-based decision support framework for the dynamic testing plan to the automated testing of client/server applications. To demonstrate how this framework works, two complementary features for testing client/server applications are illustrated in this paper.

In Section 2 of this paper, the problem of current testing tools is addressed. The application of mobile agents to the automated test execution on a 3-tier client/server application is described in Section 3. In Section 4, the concept of automated test execution through the multicast system is illustrated. Section 5 proposes a blackboard-based decision support framework for the dynamic test plan of client/server applications. Section 6 summarizes the work and offers suggestions for further study.

2. Current Testing Tools

Current testing tools with capture/playback paradigm have some limitations for client/server applications [10, 11]. Firstly, 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. Secondly, 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 [10]. For client/server applications, some test scripts for middleware can be very hard to capture/playback automatically [11], for example, the communication mechanism between clients and servers uses technology like an RPC protocol that current capture/playback tools cannot effectively capture. Finally, as a result of indeterminacy, repeated execution of a client/server application with the same test script may execute different paths and produce different results. Therefore, some mechanisms are required in order to exercise these test scripts and examine the test ordering.

To address these problems, this paper firstly introduces a flexible infrastructure for mobile agent computing: VISITOR [1], 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. These 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 banking 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 agent 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.

However, VISITOR cannot be used to solve the problem of repeated executions. Therefore, a multicast system is presented to perform repeated executions for the client/server applications. The same test data file can be broadcasted to multiple client sites to run the tests simultaneously and the test results can be returned from client sites for examining the problem of repeated executions. This approach could solve the non-reproducible problem. However, it cannot bring back the trace file from the server site, which can be achieved using VISITOR. Therefore, the dynamic testing plan advocated in this paper combines these two approaches to achieve the goal of the automated client/server testing.

3. Automated Test Execution Through Visitor

3.1. The Architecture of VISITOR

A flexible infrastructure is designed for mobile agent computing: VISITOR [1], 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. 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 architecture of the VISITOR is shown as in Figure 1 which consists of a network of agent servers, agent clients and a security server.

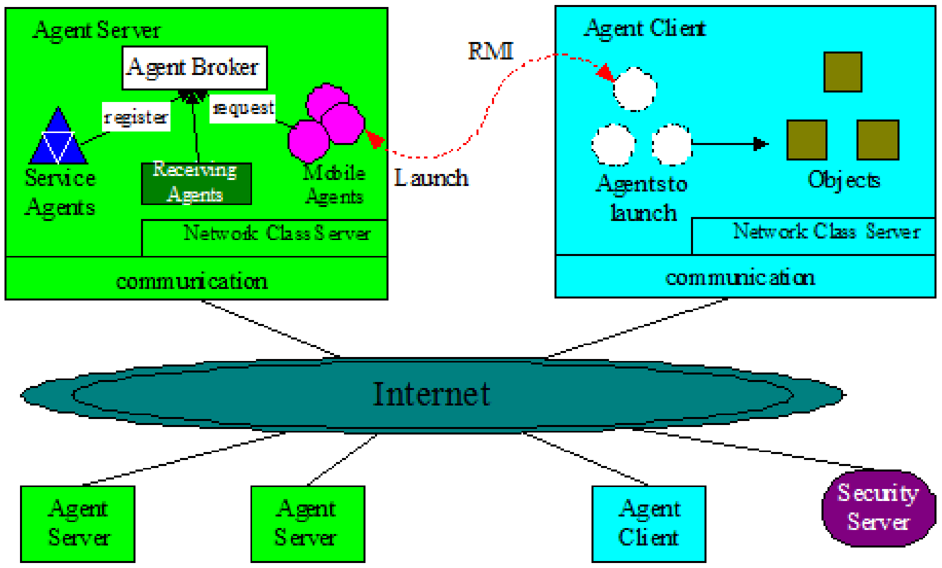

Figure 1: The architecture of VISITOR

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 accomplishing 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. In more detail for the architecture of VISITOR can be referred in [1, 3]

3.2. 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 application 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 the 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 [3].

3.3. The Application with VISITOR

The application of VISITOR to the automated test execution on the banking application is as shown in Figure 2.

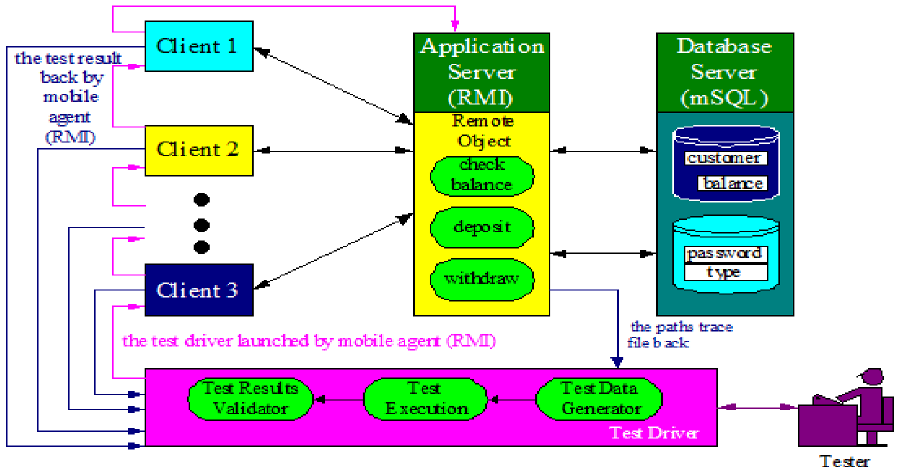

Figure 2: Automated text execution through a model agent

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 to 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 bring 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) [1,3]. 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.

The major advantages of this approach are the interaction behaviors 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. However, VISITOR cannot be used to solve the non-reproducible problem. Therefore, a multicast system is presented in the next Section to perform repeated executions for the client/server applications.

4. Automated Test Execution Through the Multicast System

A chat application allows multiple clients to enroll under a particular name and send messages to each other. A simple example of a multi-threaded client/server chat system is as shown in [3]. It makes a socket connection and then opens a window with two areas: a small input area and a large output area. After the user types the user name and the text into the input area on the client side, the text is transmitted to the server. The server echoes back everything that is sent by the user. The screen on the client site displays everything received from the server in the output area. When multiple clients connect to one server, a simple chat system is given.

The concept of the multi-threaded client/server chat system provides a direction to perform repeated executions for the client/server applications. When a client wishes to join the test, it enrolls this chat system and waits for messages from the server site. The test data file is generated and broadcast to clients on the server site. The clients receive the test data file and execute the test simultaneously and the test results are transmitted to the server. In the next Section, the concept of the multicast system is firstly introduced. The multicast framework for the client/server test execution is illustrated later.

ISSN: 2454-1907
DOI: 10.5281/zenodo.1255316

4.1. The Multicast System

This multicast system based on the client/server chat system consists of a simple chat client and a multi-threaded broadcast chat server [7]. A client class, ChatClient, implements the chat client and involves setting up a basic user interface, handling user interaction and receiving messages from the server. A multi-threaded server contains two classes: a Chat Server class and a ChatHandler class. The framework of this chat system is shown as in Figure 3.

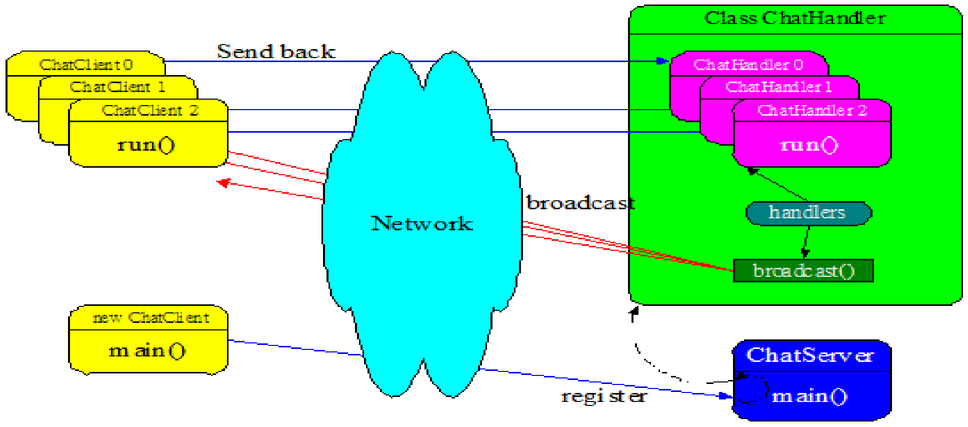

Figure 3: The framework of the multicast system

The ChatServer class is a server that accepts connections from clients and assigns them to new connection handler objects. The ChatHandler class actually does the work of listening for messages and broadcasting them to all connected clients. One thread (the main thread) handles new connections and there is a thread for each client.

4.2. An Automated Test Execution through the Multicast System

In the simple banking application, when the same test data file ran the test on two different client sites, the results were wrong as a result of the race condition on the database server [4]. This can not be tested using the mobile agents to send the same test data to two or more remote client sites to run the tests simultaneously. Therefore, the framework of automated test execution through the multicast system as shown in Figure 4 is introduced to perform repeated executions.

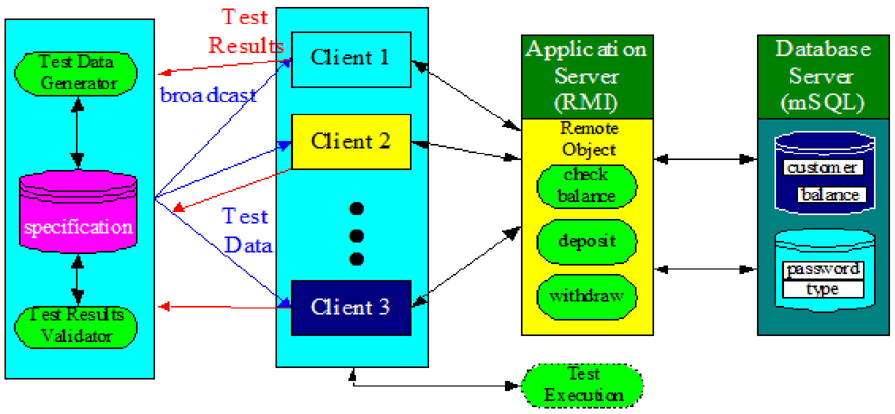

Figure 4: Automated test execution through multicast

ISSN: 2454-1907
DOI: 10.5281/zenodo.1255316

In this framework, the Chat Server class accepts connections from the clients of the simple banking application and assigns them to the connection handler objects. The Chat Handler class does the work of broadcasting the test data files which are generated from the Test Data Generator as well as the Test Execution and listening for the test results from clients to the home site for the inspection of the test results using the Test Results Validator.

Learning programming skills from some chat systems with Java RMI [12] could fit this framework as follows. To connect to a server, this framework could use the RMI name registry to get a remote reference to the Chat Server class which includes the Chat Handler class. A Client Client class could be created for the clients of application to invoke the connection to the server to request the test data file and the Test Execution. When multiple clients connect to one server, the repeated test executions could be performed.

This approach could solve the non-reproducible problem. However, it cannot bring the paths trace file from the application server, which can be achieved using mobile agents. Therefore, the dynamic testing strategy advocated here combines two approaches to achieve the goal of the automated client/server testing. The way to mix the two testing techniques is deduced from their complementary features, that is, before the multi-user test executions, a dynamic test plan to classify the testing types between multiple users on different platforms. A blackboard-based test plan for test automation is proposed in the next Section.

5. A Blackboard-Beased Decision Support For Test Automation

5.1. Blackboard Architecture

The proposed framework for the dynamic testing plan is a blackboard architecture with multiple agents. A blackboard architecture consists of a control unit, a blackboard and knowledge sources (agents) [6] as shown in Figure 5.

The control unit makes runtime decisions about the course of problem solving and is responsible for all communications amongst agents and between the user and the whole system. The blackboard is a global database containing input data, partial solutions and other data that is shared by all agents and managed by the control unit. The agents are systems that are responsible for performing certain tasks or controlling other agents.

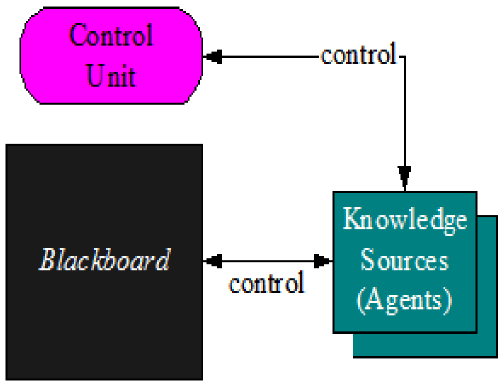

Figure 5: A blackboard architecture

ISSN: 2454-1907
DOI: 10.5281/zenodo.1255316

5.2. Illustration of the Framework

To explain how the blackboard works on the dynamic testing plan before the test execution, consider the following framework on the simple banking application as shown in Figure 6.

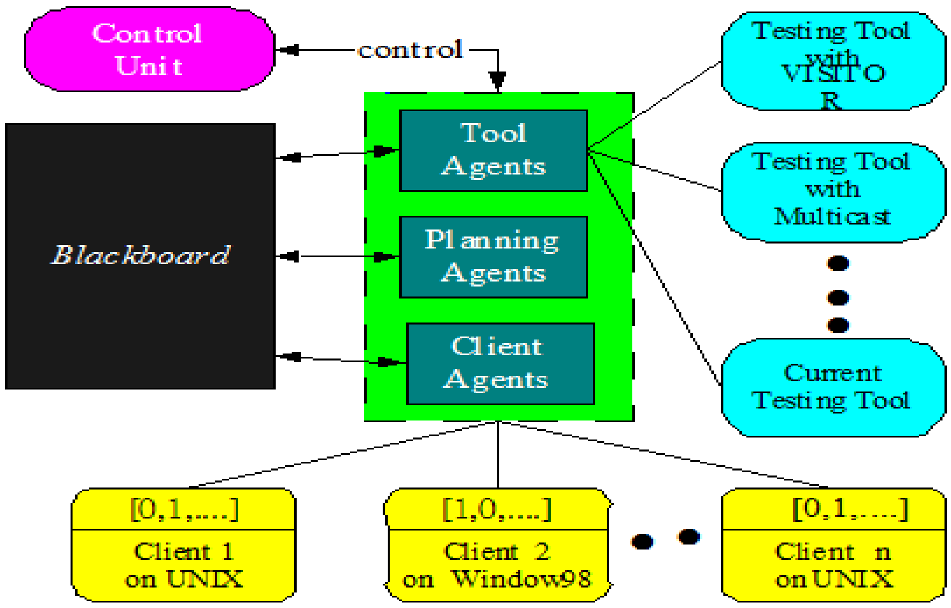

Figure 6: A blackboard-based framework for the dynamic testing plan

Assume that the global goal of testing addresses issues of indeterminacy, scalability and multi-user interaction that often determine the successful deployment of a client/server system. The initial input into the framework are testing tools with different approaches and clients will enroll the testing. The expected output is a dynamic test plan to allocate testing tasks to these testing tools to clients before the test is executed, such as which clients will use a testing tool through mobile agents.

Whenever, a new client in the client/server banking application is to be enrolled, the control unit sends a signal to the client agent to acquire the necessary information about the given client, such as what is the base of the operating system, what testing tools are suitable on this client site and what is the IP address. The client agent stores this information in the blackboard and sends a signal to the control unit so that it can accomplish its job. The control unit sends a message to the planning agent to generate a partial plan for the dynamic test execution using the information in the blackboard. In this case, the test execution using the framework of VISITOR is working effectively across platforms which are UNIX machine. Therefore, if a client under the UNIX platform wants to enroll the test, the planning agent will notice the tool agent whether or not the client is accepted for joining the testing of the multi-user interaction. The tool agent will send its decision back to the planning agent. If it is acceptable for the testing of multi-user interaction, the planning agent sends a vector which records the assignment of the testing tool to the client agent and the IP address of the given client will be added into the scripts (a visiting list for remote testing) to VISITOR through the tool agent. The planning agent sends a signal to the control unit that it has accomplished its job and waits for the coming of the next client.

The process of the dynamic test planning will be repeated until no clients join. The planning agent will check whether or not the test plan achieves the global goal for testing the client/server banking application. If so, the mission of the dynamic test planning is complete and the automated test execution will be ignited based on this test plan. Otherwise, the planning agent will re-plan (re-assign the tasks amongst clients) until the global goal is achieved.

6. Conclusions and Recommendations

To achieve software quality, software testing is an essential component in all software development. Software testing is characterized by the existence of many methods, techniques and tools, that must fit the test situation, including technical properties, goals and restrictions. There is no single ideal software testing technique for assessing software quality. Therefore, we must ensure that the testing strategy is chosen by a combination of testing techniques at the right time on the right work products. In this paper, a decision support framework based on the blackboard architecture is proposed for a dynamic test plan to mix testing approaches to the automated testing of client/server applications. This gives a reasonable proposal of the dynamic test planning for testing multi-platform client/server applications. However, undertaking the development of Multi-Agent Design Systems (MADS) has been challenging, because both theory and software support have been limited [8]. Therefore, the design and implementation of this framework will be extended in future study.

References

[1] Ahmed, S. and Nadeem, A., "Optimizing Message Delivery Cost in Mobile Agent Communication", Proc. in the IEEE Signature Conference on Computer Software & Application (COMPSAC 2012), July 2012, Izmir, Turkey.
[2] Chen, J., Greenwood, S. and Chu, H., "VISITOR: A Java-based Infrastructure for Mobile Agent Computing," Proc. in 10th International Conference on Software Engineering and Knowledge Engineering (SEKE'98), June 1998, San Francisco, USA.
[3] Chu, H. and Dobson, J.E., "Towards Quality Programming in the Automated Testing of Client/Server Applications," Proc. in 16th the Pacific Northwest Software Quality Conference (PNSQC98) and join with 8th International Conference on Software Quality (ICSQ98), October 1998, Oregon, USA.
[4] Chu, H., Dobson, J.E., Chen, J. and Greenwood, S., "The Application of Mobile Agents to Software Testing," Proc. in 15th International Conference and Exposition on Testing Computer Software (TCS'98), June 1998, Washington, D.C., USA.
[5] Corkill, D.D., "Blackboard Systems, ", AI Expert, 1991, 6(1), 40-47.
[6] Hughes, M., Huges, C., Shoffner, M. and Winslow, M., JAVA Network Programming, 1997, Manning Publication Co., Greenwich.
[7] Lander, S.E., "Issues in Multiagent Design Systems," IEEE Expert, March/April 1997, 18-26.
[8] Liggesmeyer, P., "Selecting Test Methods, Techniques, Metrics and Tools Using Systematic Decision Support," Proc. in 9th International Software Quality Week (QW96), May 1996, San Francisco, USA.
[9] Mooney, K. and Chadwick, D., "Overcoming the Challenges of Testing Client/Server Applications," Available at http://www.rational.com/support/techpapers/challenges/.
[10] Nestinger, S.S., Chen, B. and Cheng, H.H., "A Mobile Agent-Based Framework for Flexible Automation Systems," IEEE/ ASME Transactions on Mechatronics, 2010, 15(6), 942-951.

ISSN: 2454-1907
DOI: 10.5281/zenodo.1255316

[11] Quinn, S.R. and Sitaram, M., “Shrink-wrapped and custom tools ease the testing of client/server applications,” Byte, Sept. 1996, 97-102.

[12] Wutka, M., Hacking Java: The Java Professional's Resource Kit, 1997, Que Co., Indianapolis, USA.

*Corresponding author.
E-mail address: hdchu@ takming.edu.tw