AN AUTOMATED MULTI-AGENT FRAMEWORK FOR TESTING DISTRIBUTED SYSTEM

by

Ehsanul Haque

B.S., American International University Bangladesh, 2007

M.S., Southern Illinois University Carbondale, 2013

A Thesis

Submitted in Partial Fulfillment of the Requirements for the

Masters of Science in Computer Science.

Department of Computer Science

in the Graduate School

Southern Illinois University Carbondale

May, 2013

THESIS APPROVAL

AN AUTOMATED MULTI-AGENT FRAMEWORK FOR TESTING DISTRIBUTED SYSTEM

By

Ehsanul Haque

A Thesis Submitted in Partial

Fulfillment of the Requirements

for the Degree of

Masters of Science

in the field of Computer Science

Approved by:

Henry Hexmoor, PhD, Chair

Shahram Rahimi, PhD

Qiang Cheng, PhD

Graduate School

Southern Illinois University Carbondale

April 09, 2013

AN ABSTRACT OF THE THESIS OF

EHSANUL HAQUE, for the MASTERS OF SCIENCE degree in COMPUTER SCIENCE, presented on April 09, 2013, at Southern Illinois University Carbondale.

TITLE: AN AUTOMATED MULTI-AGENT FRAMEWORK FOR TESTING DISTRIBUTED SYSTEM

MAJOR PROFESSOR: Dr. Henry Hexmoor

Testing is a part of the software development life cycle (SDLC) which ensures the quality and efficiency of the software. It gives confident to the developers about the system by early detecting faults of the system. Therefore, it is considered as one of the most important part of the SDLC. Unfortunately, testing is often neglected by the developers mainly because of the time and cost of the testing process. Testing involves lots of manpower, specially for a large system, such as distributed system. On the other hand, it is more common to have bugs in a large system than a small centralized system and therefore there is no alternative of testing to find and fix the bugs. The situation gets worst if the developer follows one of the most powerful development process called continuous integration process. This is because developers need to write the test cases in each cycle of the continuous integration process which increase the development time drastically. As a result, testing often neglected for large systems. This is an alarming situation because distributed system is one of the most popular and widely accepted system in both industries and academia. Therefore, this is one of the highly pressured areas where lot of developers engaged to provide distributed software solutions. If these systems delivered to the users untested, there is a high possibility that we will end up with a lot of buggy systems every year. There are also a very few number of testing framework exist in the market for testing distributed system compared to the number of testing framework exists for traditional system. The main reason behind this is, testing a distributed system is far difficult and complex process compares to test a centralized system. Most common technique to test a centralized system is to test the middleware which might not be the case for distributed system. Unlike the traditional system, distributed system can be resides in multiple location of different corners of the world. Therefore, testing and verification of distributed systems are difficult. In addition to this, distributed systems have some basic properties such as fault tolerance, availability, concurrency, responsiveness, security, etc. which makes the testing process more complex and difficult. This research proposed a multi-agent based testing framework for distributed system where multiple agent communicate with each other and accomplish the whole testing process for a distributed system. The bullet proof idea of testing centralizes system has been reused partially to design the framework so that developers will be more comfortable to use the framework. The research also focused on the automation of testing process which will reduce the time and cost of the whole testing process and relief the developer from re-generating the same test cases over and over before each release of the application. This paper briefly described the architecture of the framework and communication process between multiple agents.

DEDICATION

To my teachers and family members.

ACKNOWLEDGMENTS

I would like to thanks specially my committee chair, Professor Dr. Henry Hexmoor. Without his guidance, encouragement and help this thesis wouldn't have been possible. My sincere thanks also goes to the committee members Dr. Shahram Rahimi and Dr. Cheng for their patience and supports. I am lucky enough to get all these nice peoples in my thesis committee. I am grateful to all my committee members for reviewing my papers several times in spite of their busy schedule. They gave several important suggestion which helps to enhance the testing framework in this stage.

I am also thankful to Dr. Mehdi Zargham and Dr. Namdar Mogharraben for their valuable suggestion about the research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Distributed systems are getting more popularity day by day. This is mainly because of high demand of processing power and request of handling large amount of data in the application. It is getting acceptance in both industries and academia as a powerful technique to design and develop software applications. There are also a number of economic and technical reasons for which distributed systems are more attractive to the software engineers than the traditional centralized systems. Some of the major factors are listed below:

- Distributed systems are cheaper than the conventional system –
    - Distributed systems are cheaper compare to the similar powerful centralized system. This is because multiple low cost commodity machines can be used to build a distributed system whereas centralized system may need special costly hardware to provide the same performance.
- Performance of the distributed systems are better than the traditional system –
    - Performances are far better than the centralized system as distributed system combines the processing power and storage capacity of the multiple machines.

- Distributed systems are scalable –
    - Processing power and storage capability of a distributed system can be increased easily by adding more nodes to it.
- Distributed systems are reliable –
    - This is one of the most important features of the distributed system. Using redundant computers and/or hardwares, faults of hardware or software can be reduced drastically. Therefore failures of one component will not shutdown the system completely.
- Distributed system can be available to a group of people –
    - Because of its distributed capabilities, different server can serve different group of people based on their taste and needs whereas others might have different. Therefore it is possible to deliver some popular contents to a specific group quickly.

Distributed systems and grid computing system have some similarities but there are some clear separations between them. Grid computing mainly used to increase the performance by combining a large number of processors for a particular task. On the other hand, a machine in distributed system can runs varieties of software. Therefore the applications of distributed systems are more comparing to grid computing system. Distributed systems are also dynamic and can change behavior over the time and based on the environment. As a result, testing distributed systems are far complex and time consuming process. Besides these factors, there are also issues in distributed systems like concurrency, fault tolerance, security, interoperability, etc. which makes difficult the testing process.

1.1 COMMON TECHNIQUE OF TESTING DISTRIBUTED SYSTEM

Most of the time distributed system engineers are worked under extreme pressure. This is because of the popularity and complexity of the distributed system. There is a very high demand of the distributed system in the market and this demand is increasing every day. Because of this extra pressure, distributed system engineers are sometimes forced to do a sloppy testing process and ends up with a low quality product. Another main reason is the lack of efficient and user friendly testing framework for distributed system. There are several procedures are being practiced by distributed system engineers to test a distributed system. Some of the common procedures are listed below:

1.1.1 TEST USING SIMULATION

This is probably one of the most common techniques of testing distributed system. This is popular mainly because this technique does not need the actual infrastructure of the distributed system to test the system. Therefore, it saves a lot of time and cost of the developer. However, this technique often cannot describe the actual behavior of the distributed system. In this technique, developer predicts the actual behavior and the environment of a distributed system and based on those assumptions they create a model and simulate the system. This type of testing might be helpful for the system that works on some specific formulas (e.g. electronic circuit or networks) or system that follow the law of physics. There are three major problems of simulation based testing:

- It is often difficult to predict the environment of the actual distributed system using simulation since distributed systems are dynamic.
- Analyzing performance of the actual distributed system is very difficult using the simulation technique because there might have lot of other factors which is sometimes difficult to address during the simulation.
- Actual communication cost is not possible to predict using this technique as the testing occurs on a virtual model in simulation based testing.

1.1.2 MIDDLEWARE BASED TESTING

This is one of the most common and effective technique of testing centralized system. Middleware connects the presentation layer and database layer. It receives the request from user through presentation layer and then process data with the help of data layer. Finally it sends back the processed data to the presentation layer which displays in the user's screen. However, this technique is also not very helpful to test distributed system. The major reason for why this is not an effective testing technique for distributed system is-

- Testing middleware do not guarantee some of the common properties of distributed systems such as concurrency, fault tolerance, etc.

1.1.3 TEST USING EQUATION

In this method, the operation of a distributed system is described using an equation/formula. Unfortunately, this is also not a very effective method to test a distributed system. The major problem of testing using equation is, it uses some assumptions to prepare the equation. That assumption could be wrong for some of the distributed systems. Also it is very difficult to predict the environment for some distributed system. Therefore test using equation might be helpful to test some particular distributed system but it is often failed to generalize the testing process for other types of distributed system.

1.1.4 TEST UNDER REAL ENVIRONMENT

This is another effective method of testing distributed system. This type of testing works on the actual distributed system and therefore capable of find out the actual bugs of the system. However this type of testing requires setting up the actual distributed environment before the test taking place so that it can work on the actual behavior of the distributed system. Therefore this technique is a bit expensive technique but saves a lot of time by early discovering bugs in the system. Also it takes longer time to complete the test cases compare to the simulation based technique.

1.1.5 TEST UNDER CONTROLLED ENVIRONMENT

This is another acceptable method of testing distributed system. In this technique developers mimic the actual hardware of the distributed system. Therefore it saves the cost of testing actual system. Testing under controlled environment is referred as emulation based testing. In this technique, developers can control the environment of the system such as increase/decrease the workload of the distributed system, specify the memory size of the machine, etc. There are some software commonly used to accomplish this type of technique such as VMWare, VPC, etc. Microsoft Visual studio also has a great number of testing tools where software engineers can control the number of connection, different types of browser, number of processor, memory size, etc. during the testing which is very helpful.

1.2 DISTRIBUTED SYSTEM TESTING SOFTWARE

There are some software already exist in the market to test distributed system. Most of them came with several restrictions. Some of them focus on specific technology/domain therefore it cannot be used in other platform/domain (e.g. some of them only focusing on BIZTalk solutions). However- the main purpose of the generic test framework for distributed system is to be able to swap a distributed system with other and still have the capability of generating and executing the test cases and display the test results in a standard way. There are also some software focused on specific types of testing (e.g. load testing, integration testing, etc.). Though we focused on the unit testing of the distributed system in this research but our target is to design the framework such a way so that the logic of the testing can be easily extend to other major types of testing in future.

1.3 THESIS OUTLINE

The remaining of the paper is structured as follows-

- Chapter 2 is discussion about related works and literature review
- Chapter 3 covers the system design of the test framework including the structure of the framework as well as the interaction between different agents.
- Chapter 4 is discussion about the evaluation of the framework
- Chapter 5 discuss the conclusion reached and future work required

CHAPTER 2

RELATED WORK AND LITERATURE REVIEW

The overall view of testing process has been changed a lot during last couple of years. Software engineers now consider testing is a part of the development and maintenance activity. They are no longer wanted to consider testing as a post activity. Previously, software engineers used to test the system after completion of coding phase (e.g. Waterfall model has five phases and testing takes place right after the implementation phase). Therefore, there is a huge change in software development life cycle/ process. It also indicates that software companies are giving more importance on testing now a day than before which is a very good sign for the whole software industry as it leads to develop quality software. Previously, most of the software companies followed waterfall model or spiral model or iterative or incremental model etc. But these days most of the software companies following either core agile development or the variations of agile development model (e.g. extreme programming development). Agile development model is an iterative and incremental development model. One of the major differences between agile development model and traditional model, like waterfall model, is the testing process. In agile development, there is a testing phase in each cycle and the duration of each cycle is around 1-4 weeks. Therefore, testing happens at different parts of software development life cycle in agile development. This technique is also referred as continuous integration process. On the other hand, as we mentioned above, in waterfall model testing happens after completing the system development. Testing in each cycle is very important. It saves time and labor cost drastically because if any glitch/error detect at the middle of the development in waterfall model, developer needs to go back to the beginning and write new code, all over again which is not the case for agile development. In agile methodology work is evaluated at the end of each cycle. Agile development ensures the developers that bugs are detected in the development cycle and product is double checked after the removal of the bug which gives more confident to the developer about the completed modules and the whole system. However- there is a major drawback of this process. Developer needs to spend a lot of time to write the test cases in agile development as testing happen in each cycle. Also, since this is a manual process, most of the time developers need to re-write the test cases once the source code updated. Still this process is tolerable for traditional system as there are many open source and commercial testing frameworks exists for the traditional system but unfortunately there are very few for distributed system. It’s a big pain to rewrite the test cases for a large distributed system in each cycle. Therefore we really need an automated process to test the distributed system in each cycle of the development. In the rest of the chapter, existing testing techniques and related works on testing distributed system has been discussed briefly.

2.1 SOFTWARE TESTING

In a simple word, software testing verifies a particular program or module or section meets the project requirements or not. Some people compare testing with debugging. But testing is more than just debugging a program. Software testing can be a requirement or acceptance testing, system testing, integration testing, unit testing, reliability estimation testing, load testing, etc. A formal definition of software testing is-

Software testing is the process that validate and verify a software: [1]:

1. Meets the requirements of its design document.
2. Works as indicated in the requirements.
3. Can be implemented with the same characteristics.
4. Satisfies the needs of the users.

There are two major classification of testing process. They are (1) Functional testing and (2) Structural testing. There is a very short description of these two types of testing in next section of this paper for the reference.

2.1.1 FUNCTIONAL TESTING

Functional testing is referred as black box testing. In functional testing, functions are tested normally by providing input to the functions and compare the output of the functions with the expected results. There is no need of knowledge about the internal structure of the system for functional testing. Functional testing basically checks the overall behavior of the program. Functional testing basically involves the following five steps[2]:

1. Select the functions which are expected to perform.
2. Generate input data based on the requirements.
3. Find out the expected result based on the requirements.
4. Run the test case.

5. Finally compare the actual result with the expected result.

Most common functional testing are: system testing, regression testing, user acceptance testing, etc. This kind of testing is done by mostly non-technical users as they do not need knowledge about the source code to perform the testing. These types of testing do not guarantee that the system will perform as expected for every condition. Therefore, we need to test the internal structure of the system. Here structural testing comes in the scene.

2.1.2 STRUCTURAL TESTING

Structural testing is also known as white box testing. To perform structural testing, tester must to have knowledge about the internal logic of the system. The main purpose of structural testing is to make sure that the functionality of the system is working as described in the specification. Most common structural testing are: path testing, code coverage testing and analysis, logic testing, unit testing, load testing, stress testing, performance testing, security testing, etc. Structural testing techniques include[3]:

- Control flow testing
- Data flow testing
- Branch testing
- Path testing
- Statement coverage
- Decision coverage

This paper focuses specially on the structural testing, more specifically unit testing. Software engineers often prefer to perform structural testing because of its advantages. However- structural testing is complex and difficult compare to the black box testing. The major two difficulties to perform structural testing are: [4]

1. Testers have to have full knowledge about the system and source code to do the structural testing.
2. Sometimes it might not possible to test every single conditions of a function. Therefore, it is certainly possible to skip or remain untested some conditions.

In this research we have tried to overcome the above difficulties by proposing an automated testing framework. More details of the framework have been discussed on chapter 3.

2.2 DIFFERENT TESTING ACTIVITIES

Each phase of the software development life cycle can be test using different level/types of testing. Each of these testing focused on specific types of faults/bugs of the program. Some of the major testing technique and their scope are listed below in Table 1.

Table 1: Different testing activities

Name of Testing	**Description**
1. Acceptance Testing	This is a high level testing technique which is normally created by communicating with client, tester, business analyst and developer. It is often expressed in a business domain language. This type of testing normally performs right before the delivery of the product to ensure that the product met its requirement.
2. System Testing	System testing is referred as black box testing. It's not only evaluates a particular area of the system but it evaluates the whole system with the system requirements.
3. Integration Testing	This is another popular testing technique which detects any faults between the software or hardware that integrated together. The target of the integration testing is to verify the performances and/or functional and reliability requirements according to the specification.
4. Unit Testing	Unit testing is probably most popular testing technique among all other testing which checks that a particular section or module or unit meets the specific requirements or not. A unit can be a method, a class or a module.

2.3 CHALLENGES OF TESTING DISTRIBUTED SYSTEM

A distributed system is a group of independent machine or nodes that works as a single system to its end user [5]. Usually distributed systems are much larger and powerful compare to a traditional system. This is because it combines the capabilities of distributed software/hardware. A perfect distributed system should have the following characteristics[6]:

- Fault-Tolerant: It should recover when hardware fails rather than shutdown or perform an incorrect action.
- Highly Available: It can execute operations, provides services even some of the components failed to perform.
- Recoverable: Components should restart themselves and refresh the system automatically after a failure happen.
- Consistent: It doesn't matter how many machines communicates with each other but they should acts like a single system and coordinate actions by multiple distributed components.
- Scalable: It should be scalable and virtually there will not have any significant effect of adding more nodes to a distributed system. It should operate the system perfectly as it was intended.
- Responsive: It should be responsive and therefore always response in a timely manner.
- Secure: It should authenticate the access to data and services.

There are several challenges of testing distributed systems. One of the most challenging issue is- a centralized system resides on a single computer whereas a distributed system can be resides on multiple locations of the world. Hence testing and verification of distributed systems are far difficult compare to a centralized system as testing involves all parts of the distributed system. Besides this, there are also issues like fault tolerance, availability, concurrency, responsiveness, security, etc. which makes the job more complex and difficult. One of the important characteristics of the distributed system is responsiveness. So if any of the distributed components fails to response then how long to wait to decide that the test failed is another critical issue. Failed to response doesn't always mean that the system is buggy. It can happen for many reasons. One of the common reasons is the network communication problem or may be because of hardware failure. Another important issue is distributed server can have different platform and/or architecture. As a result, there is a chance of data loss or serialization problem since different platform handle same object differently which is often refer as boxing/unboxing. Testing framework should have the ability to find out which server cause the problem. There can have communication problem between different platforms too. Different server can use different protocol for the communication. Therefore it can create a serious problem to test the system. Finally, security can be another concern. Messages should not pass across the public network which can cause data lick.

We have designed a distributed chat application to simulate the testing framework proposed in this paper and addressed the above issues. Chat application uses a central server to keep all the messages from different users. It is a client-server application where the server takes the request from different clients and processes the request with the help of database server and finally sends back the result to the client. The server uses soap protocol for the communication between server and client. Therefore it supports clients from different platforms (eg. .NET, Java, Python, etc.). Client machines ping continuously after a certain interval to check updates from the server. The whole process illustrate in Figure 1 from the bird's eye view.

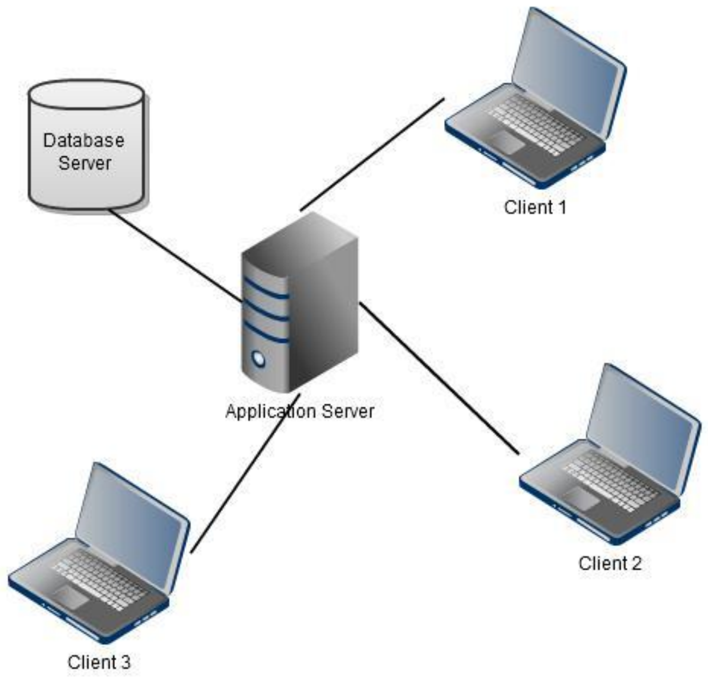


Figure 1: Communication between client and server in a distributed chat application

2.4 BENEFIT OF MULTI-AGENT BASED SYSTEM

Now a days multi-agent based system are more popular for modeling, designing and developing software systems. Also it is getting acceptance widely as there are a good number of development platforms, tools, modeling languages and programming languages exists for multi-agent systems. One of the major benefits of multi-agent systems is- in multi-agent system, agents are autonomous. They act autonomously on behalf of users in distributed environments. Therefore, agent can do their task independently by communicating and sharing information with other agents. This property of multi-agent system is really very helpful to design testing framework for distributed system. There are two types of agent exists in multi-agent system. They are local and global agents. Local agents do not have a global view of the system. Therefore local agents communicate with other agents and collect the required information to accomplish the task assigned to them. Agents also update their knowledge from the environment. In multi-agent system, agents are specially used to solve complex problem that are quite difficult to solve by a single agent. Therefore, multi-agent system provides more natural way to represent a problem. Using multi-agent system, it is easy to represent team planning, task allocation, progress report, etc. Multi-agent provides reliability, maintainability, robustness, efficiency, reusability, flexibility, etc. It also enhanced the overall system performances. Multi agent systems are decentralized and do not have the “Single point of failure” problem like the centralized system. Multi-agent systems are already being using in multiple different domains successfully such as aircraft maintenance, wireless collaboration and communication, military logistic planning, supply-chain management and many other areas [7]. Because of all these wonderful properties of multi-agent system, it could be a very good and more realistic platform to design and develop a user friendly testing framework for distributed system.

2.5 RELATED WORK

There are number of people already realized the importance of testing distributed system and started research on this issue. Different authors worked on different issues of distributed system. Some of them worked on the major properties of the distributed systems such as concurrency, security, timing, reliability etc. [8, 9]. Some of the authors did a very good improvement of the testing process[10, 11]. There are few authors did research out the box to get success. For example, some of the authors did research on random testing technique to test a distributed system [12, 13]. Unfortunately, it might not give guaranteed performance in all environments. All these novel work has been considered carefully to design the testing framework proposed in this paper. There are also a group of authors believe that model checking can be a best and most effective technique to test distributed system. Some of the researcher also votes for testing middleware only in case of testing distributed system[14]. They assume that the middleware is capable of handling all concurrency and fault-tolerance issues. This seems very ambitious for testing distributed system. However- this is a proven and common technique to test a centralized system. Besides these, there are also a number of proposals are exists to test concurrent and distributed systems. However, most cases it is not clear enough that how these techniques can be scale up or how widely and on what conditions it can be applicable to the real world distributed systems. However- one author named Dowling [15] presents a promising method of testing distributed system for real time Ada system. According to Dowling's theory, below are the basic steps of testing distributed systems:

1. Test the internal logic of a unit on the host, using dummy units to send or receive message to or from the unit of interest.
2. Test the unit on the host when it interacts with real version of other units executing with pseudo parallelism.
3. Test the unit when they are distributed with true parallelism.

Dowling also mentioned that a special tool can be used to generate the stubs for both client distribution object and server distribution object. These stubs communicate using the underlying infrastructure of communication service. This research indicates that we can partially reuse the concept of testing traditional system to get a user friendly testing framework for distributed system. System design of the proposed testing framework has been described briefly in the next chapter.

CHAPTER 3

SYSTEM DESIGN

This model is inspired by the hierarchical organizational structure. An organizational structure consists of different activities. Among them, some of the major activities are task distribution, communication between different hierarchies and supervision which helps to achieve a specific goal. It describes which individuals need to participate in which decision process. There are two major organizational structure exists in most of the organizations. They are Flat structure and hierarchical structure. In flat organizational structure, there are very few or no level of management between management and staff level employees. This structure is especially very effective for small organization. The major benefit of this structure is, speeding up the coordination and communication between management and employees. On the other hand, there is another popular organizational structure exists which is suitable for medium to big organization. Often it called as hierarchical organizational structure. Our model can be compared with the hierarchical organizational structure of an organization. In this chapter we describe how this framework is comparable with an hierarchical organizational structure and the design of the framework briefly.

3.1 ORGANIZATIONAL STRUCTURE VS TEST FRAMEWORK

We consider an IT (Information Technology) company as an example for our model. We observed how an organization functions by different categories of employees such as CEO, CTO, Team Manager, Sr. Software Engineer, Junior Software Engineer, etc. We also observed that the task distribution process between the different individuals, their communication process and the supervision in a hierarchical organizational structure. Hierarchical structure follows the layout of the tree. There will be several departments/groups in this structure that are responsible for different module of the project and each department/group will have a team leader who communicates with team members and upper management often called as team leader. Team leaders ensure that the specific module which is assigned to the team, meets the client's requirements. In this structure, employees narrow down their focus on specific tasks and become specialists on a specific area. For example, in a small IT company, we find that there are several team and each team have specialists such as software analysts, database experts, software engineer, UI(User Interface) engineer, testing engineer, etc. Communications between different departments are very important in this structure as each department have knowledge on what has been assigned to them rather than knowledge on the organization as a whole. In addition to this there will be competition between different employees in hierarchical structure as it will leads them for the opportunities of the promotions and better salaries. Task distribution is also different in hierarchical organizational structure. Team leader distribute the task among team members when they start working on a project. Team leaders are the bridges between the team members and upper management. They communicate with the upper level management about the progress of the project. One person will have all the updates at the end of the day from the entire department often called CTO (Chief Technical Officer)/Project Manager. CTO’s role is to assure the successful execution of the project’s requirement, risk analysis and deployment. They often communicate with the client and play role as a messenger to the team regarding the project status. There is one highest position in most of the organization often refer as CEO (Chief executive officer). CTO and CEO communicate with each other and get overall impression about the project and team. CEO deals with the agreement of the project and gives a high level overview of the project to client. CEO represents the whole company in front of the world and attends in important business meetings. These hierarchies may differ from company to company. There can be more hierarchies in some companies according to the need and types of project they are dealing with. But the overall procedure is more or less same for most of the companies. Figure 2 shows a top level view of an organizational hierarchy.

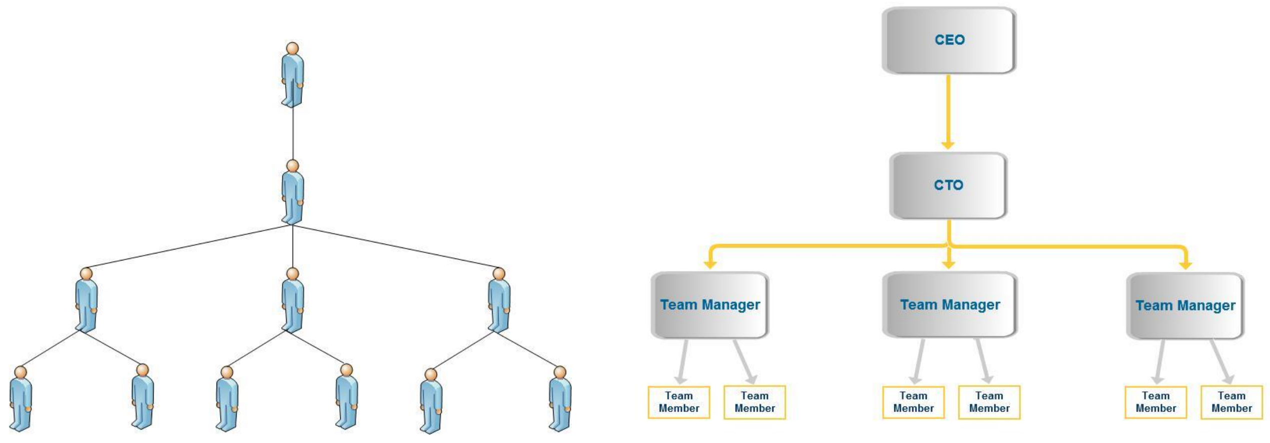


Figure 2: Top level view of an organizational hierarchy

If we analysis the whole system closely from the beginning to the end of the project then we will see there is a very good message passing system between different hierarchies of people in hierarchical organizational structure. We did research on this organizational model and found that this idea is a well fit in software to test a large distributed system. Different hierarchies of this model can be replaced with different agent and they can collaborate with each other to accomplish the whole testing process. Rest of this chapter, we describe the system design of the test framework.

3.2 SYSTEM DESCRIPTION

There are three major components in the framework: 1) Test Engine, 2) Test Generator and 3) Test Agent. Test Engine mainly distributes the task between different agents and collects result from them once they are done. Therefore Test Engine can be compared with the CTO of the organization who also distributes the task between employees and prepare a progress report for projects. Test generator generates the test cases and group them based on the configuration file. Each group can have one or more test cases. This specially helps to find defective module quickly at the end of test. Test generator works by collaborating with Test Engine. As soon as it receive a command from Test Engine and get all required information about the system, it starts generating test cases for the system. There are multiple local and distributed Test Agents which works on the test cases. These agents have special access to the system as they deal with the dynamic environment of the distributed system. The details about this will be provided soon. Test Agents works independently and interact with each other to accomplish the assigned task. Test agents do not have knowledge on the full system. Groups of Test Agent work together on a test module which may contain multiple test cases. Hence this group of agent can be compared with “team” of the organization where one of the agents will play the team leader role. The whole framework can be customized by a configuration file which is basically a XML file. Figure 3 shows the major components of the framework and the communication between them.

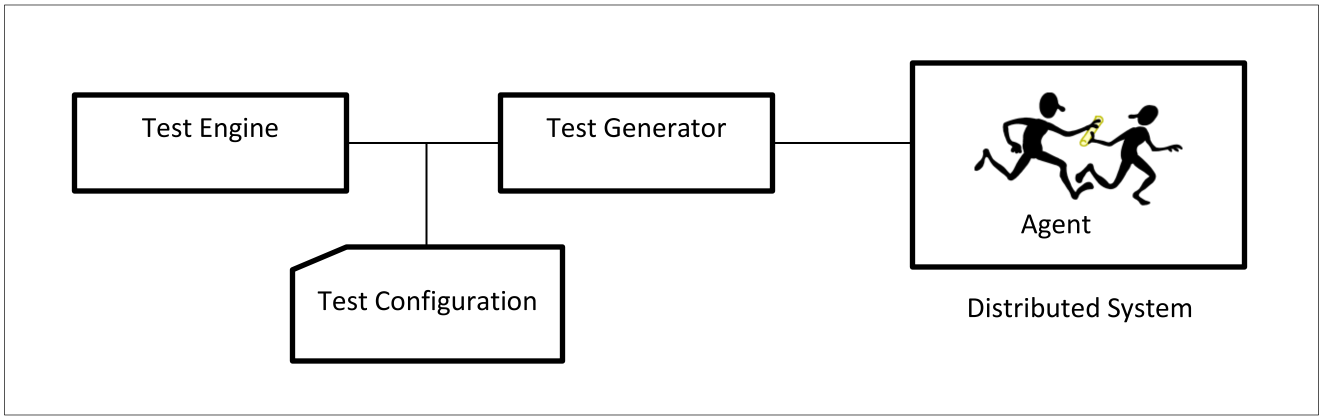


Figure 3: Major components of the framework

We will discuss on each of these components briefly in next section.

3.2.1 TEST ENGINE

Test engine is the test initiator as well as the test terminator. Test engine setup the testing environment based on a configuration file called test configuration file which is basically a simple xml file. Test configuration file contains very important information regarding the test environment such as connection end points (Address, contract and binding information) of distributed server, application variables (mock object framework configuration, random value generation logic), etc. Below are the major responsibilities of the Test Engine–

a. Initialize the test environment based on the test configuration file
b. Communicate with different test agent to reveal the structure of different distributed module. Test agents can be resides in different distributed server. Since Test Agents have local access to the server, they reveal the application

structure using the reflection technique. Communication between different Test Agent handled by Microsoft Windows Communication Foundation (WCF) framework. The major benefit of using WCF is it supports a number of transport schemas besides the popular HTTP binding like TCP, Peer Network, Named Pipe, MSMQ(Microsoft Message Queue), WSHTTP, and WebHTTP (used for restful service) binding. Therefore, if different server uses different technology, this framework still works.

c. Send signal to "Test Generator" to generate the test cases.
d. Test Engine validates the test cases before execute in next step.
e. Start the testing process by sending a signal to the test agent in different server. It also provides the relevant test cases to the test agents.
f. Test Engine collects test result from test agents once it completes the test cases and save the result in the physical disk.
g. Display and analysis the test result.
h. Terminate testing process.

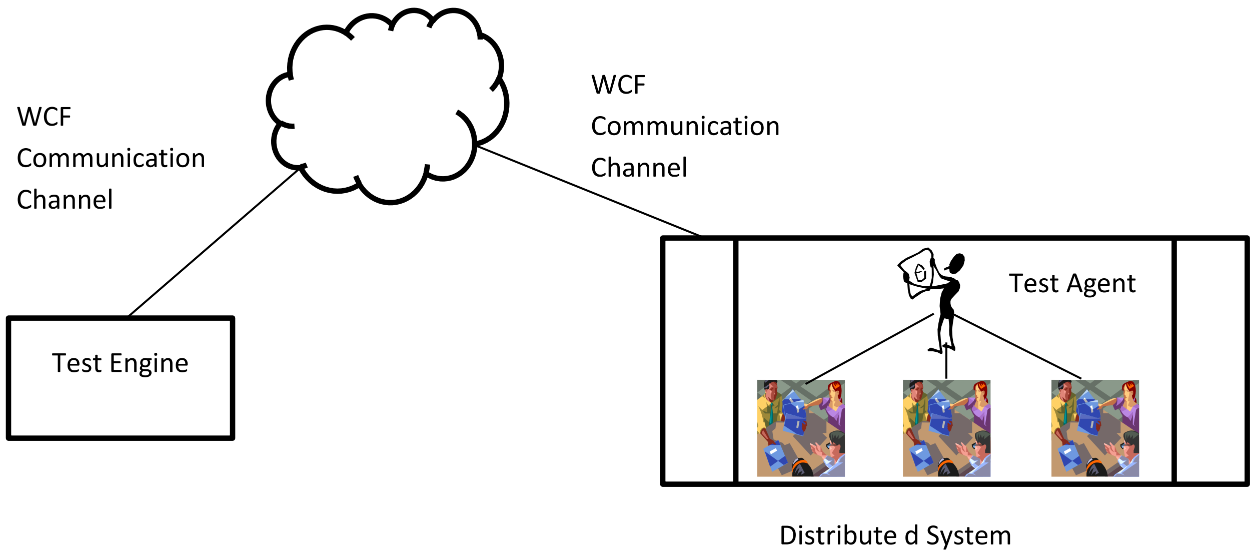


Figure 4: Task distribution and communication of the system

3.2.2 TEST GENERATOR

Test generator made the testing process automated. Test generator is an independent agent. It accepts request from Test Engine. It also receives the structure of the application from Test Engine that need to be test. Test generator reads the structure of the application and generates test cases using mock framework. Finally, it sends back the test cases to the Test Engine. Test Engine sends the tasks to the test agents for test. So below are the lists of the responsibilities of the Test Generator:

a) Accept test generation request from test generator
b) Generate test cases with arguments list based on the application structure. Test Generator reads the mock configuration from the Test Configuration File.

3.2.3 TEST AGENT

Test agents are independent agents that received command from Test Engine and accomplished the task. Test agent creates sub-test agents to test a Test Module. Each test module may contain multiple test cases. Test agents are multi-threaded. Therefore if one test case takes long time to complete, it can work on other test cases in parallel rather than waiting. When sub test agent finished the execution, it submits the result to the parent Test Agent. Here parent Test Agent can be compare with the team leader role in an organizational structure and sub test agent as team member. Test agents are local to the server. Therefore they have full access to the resources. Below are the major responsibilities of a Test Agent:

a. Exposes class, methods and properties definition from assembly using reflection technique and send back the result to the Test Engine.
b. Test agents are multi-threaded so that it creates sub agent for each test cases. Maximum number of sub-agent creation can be handled by test configuration file.
c. Sub-Test Agent completes the test case and sends back the result to the parent Test Agent. Once parent Test Agent get result for all test cases of a test module, it sends the result to Test Engine.

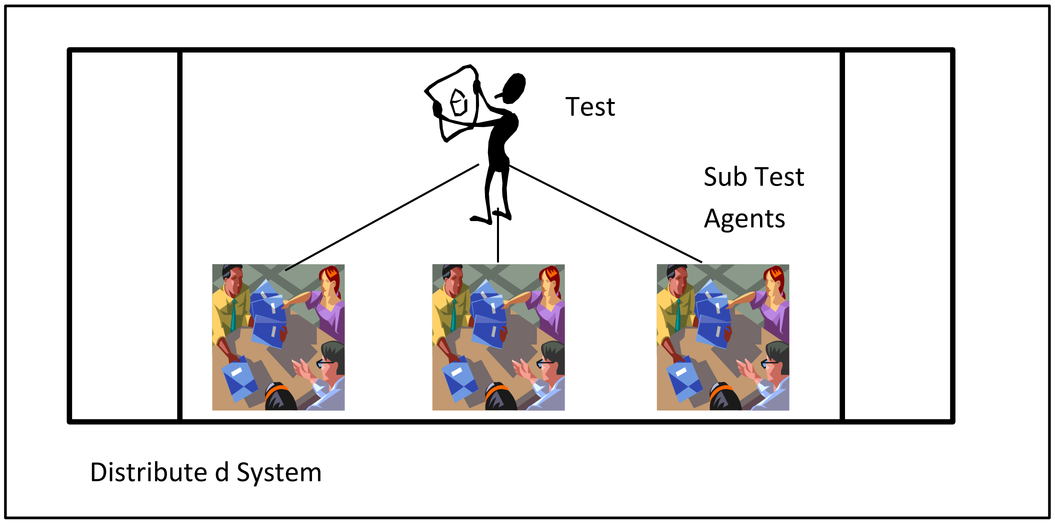
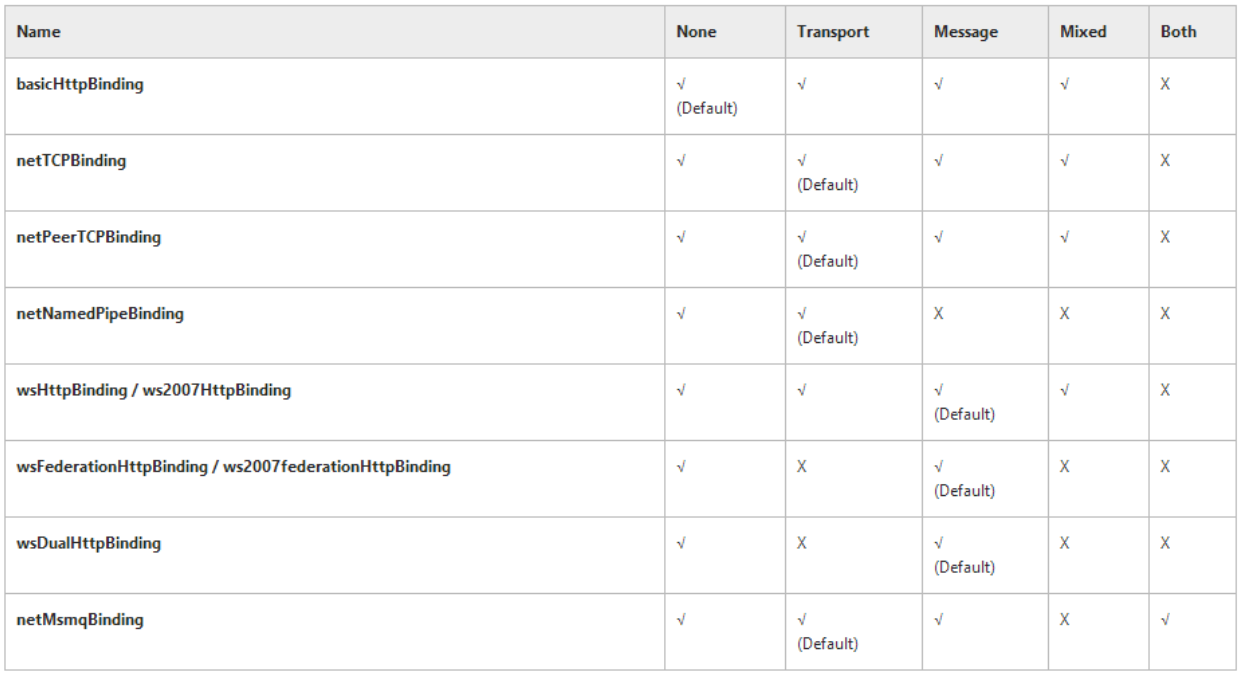


Figure 5: Task distribution among test agents

3.2.4 TEST CONFIGURATION FILE

Test configuration file is basically a XML file which contains the server information as well as application variable. Test Engine reads the test configuration file and set the test environment based on that. Figure **6** shows a sample Test Configuration file. As stated above, WCF (Windows Communication Foundation) framework used to communicate with Test Agent in different server. Therefore we need to set behavior services and binding information in the configuration file. Different distributed server should be listed in the services tag. Each service has endpoint which defines the Address, Binding and contract information of the service. In short it is define as ABC (Address, Binding, and Contract). Table 2 shows list of binding supports in WCF and the transfer security modes of the bindings[16].

Table 2: List of bindings supports in WCF and transport security modes of the bindings



Name	None	Transport	Message	Mixed	Both
basicHttpBinding	√ (Default)	√	√	√	X
netTCPBinding	√	√ (Default)	√	√	X
netPeerTCPBinding	√	√ (Default)	√	√	X
netNamedPipeBinding	√	√ (Default)	X	X	X
wsHttpBinding / ws2007HttpBinding	√	√	√ (Default)	√	X
wsFederationHttpBinding / ws2007federationHttpBinding	√	X	√ (Default)	X	X
wsDualHttpBinding	√	X	√ (Default)	X	X
netMsmqBinding	√	√ (Default)	√	X	√

Besides WCF configuration, the Test Configuration file also contains TestGenerator, TestAgent and ResultSet tag. TestGenerator contains the mock framework settings. Test Agent specific setting will be inside the Test Agent tag such as Masimum number of thread for each Test Agent, Maximum execution time for a test case, etc. Resultset tag allows customizing the display of the test results. It helps to analyze the test result and quickly find out the bugs in the system.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<TestConfiguration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="Service1Bevhavior"/>
        <behavior name="Service2Bevhavior"/>
      </serviceBehaviors>
    </behaviors>

    <services>
      <!--SERVICE ONE-->
      <service name="Service1">
        <endpoint address=""
        binding="netTcpBinding"
        bindingConfiguration="tcpServiceEndPoint"
        contract="ListenerService.IListenerService"
        name="tcpServiceEndPoint" />
      </service>

      <service name="Service2">
        <endpoint address=""
                            binding="netTcpBinding"
                            contract="UploadObjects.IResponseService"
                            bindingConfiguration="TransactedBinding"
                            name="UploadObjects.ResponseService"/>
      </service>
    </services>

    <bindings>
      <netTcpBinding>
        <binding name="tcpServiceEndPoint" closeTimeout="00:01:00"
                openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
                transactionFlow="false" transferMode="Buffered"  transactionProtocol="OleTransactions"
                hostNameComparisonMode="StrongWildcard" listenBacklog="10" maxBufferPoolSize="524288"
                maxBufferSize="65536" maxConnections="10" maxReceivedMessageSize="65536">
          <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
           maxBytesPerRead="4096" maxNameTableCharCount="16384" />
          <reliableSession ordered="true" inactivityTimeout="00:05:00"
           enabled="true" />
          <security mode="None">
            <transport clientCredentialType="Windows" protectionLevel="EncryptAndSign" />
            <message clientCredentialType="Windows" />
          </security>
        </binding>
      </netTcpBinding>

      <netTcpBinding>
        <binding name="TransactedBinding">
          <security mode="None" />
        </binding>
      </netTcpBinding>
    </bindings>
  </system.serviceModel>

  <TestGenerator>
    <!-- Mock Framework Configuration Settings-->
    <mock>
      <framework name="" assembly=""></framework>
    </mock>
    ....
  </TestGenerator>

  <Testagent>
    <!--Settings for Test Agent-->
    <add name="MaxThread" value="4"></add>
    <add name="MaxExecutionTime" value="10"></add>
    ...
  </Testagent>
  <Resultset>
    <DisplaySummary>True</DisplaySummary>
    <AllowGrouping>True</AllowGrouping>
    ....
  </Resultset>
</TestConfiguration>
```

Figure 6: Sample test configuration file

3.3 AGENT INTERACTION

Figure **7** shows the sequence diagram of the system.

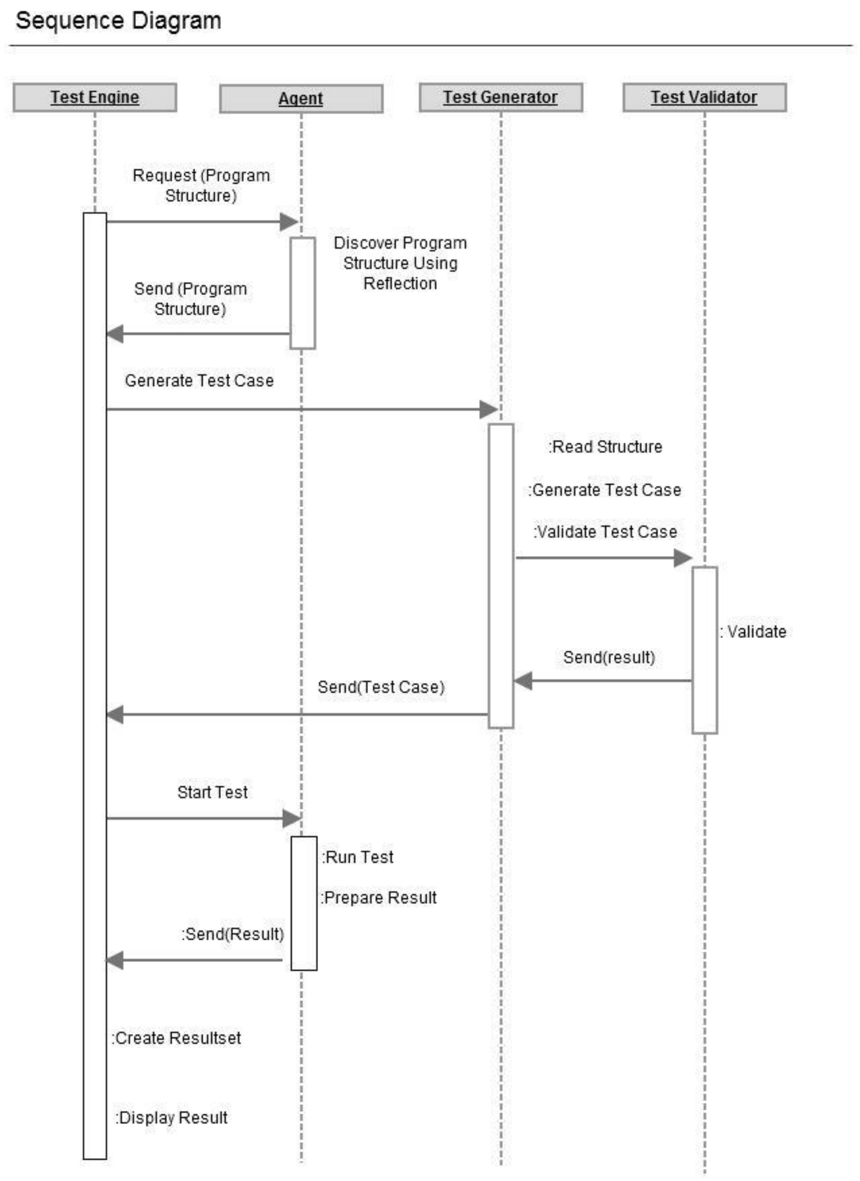


Figure 7: Sequence diagram of agent interaction in the system

From the above diagram we are seeing that Test Engine first start the program by requesting program structure to the Test Agent. Test agent discovers the application structure using reflection method and sends back the result to the Test Engine for futher processing. This process makes the system automated. Test Engine always gets the up to date program structure of the application. Therefore, Programmers do not need to update the program structure manually after each modification of the system. It also eliminates writing extra coding. Test Generator is another agent which works next with the program structure and generates test cases based on that. Test generator's life line starts when it gets the command from the Test Engine to generate test cases. Finally, it validates the test cases using Test Validator agent and sends back the test cases to the Test Engine. Now Test Engine has all the information to run the test. It communicates with the Test Agents to start the testing process next. Test agents receive a command from Test Engine with a list of Test Cases. Test agent then runt the test cases with help of other local agents and send back the result to the Test Engine. Finally, Test Engine creates the result set once it get all the test result from Test Agents and displays the result to the user. Test Engine allows user to customize the display screen so that programmers can find a specific issues quickly as well as can use the screen in the report.

3.4 RESULTSET

After finishing the execution of test cases, Test Agents collects the result from its sub-agents. Finally, it sends the result to the Test Engine for further processing. Test Engine stores all the test result to a result set XML file.

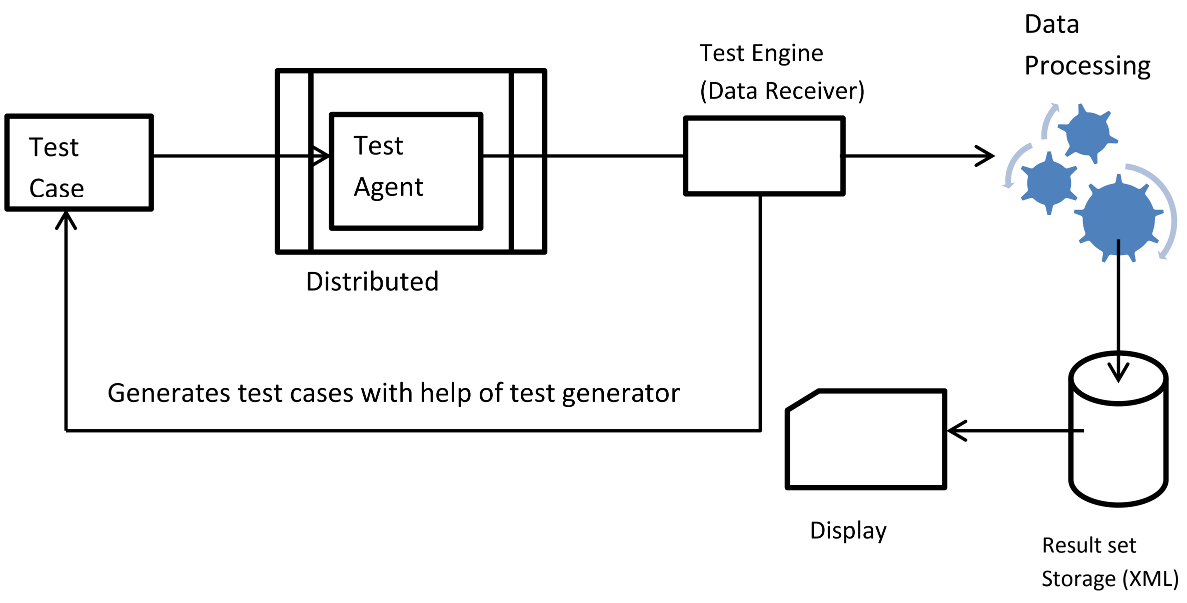
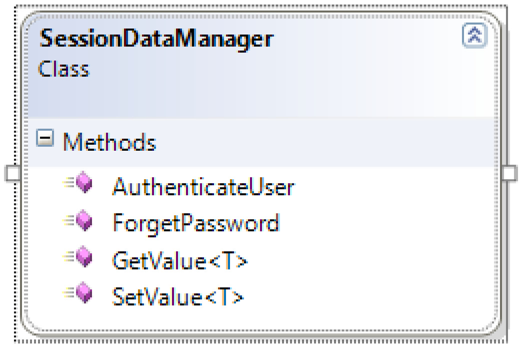


Figure 8: Process of data collection for result set

Result set XML is a well-structured XML file where test cases are grouped by Test Module. Each of this group can have multiple test cases. It also allows user to customize the groups. Main benefit of using this approach is we can see test result for individual test cases as well as the whole group at a glance. For example, say we have a class SessionDataManager where SessionDataManager have four methods. They are:

- GetValue(string key) – Returns session object based on the key user provides.
- SetValue(string key, string value)- Stores object in application session against the user provided key.
- AuthenticateUser(string username, string password)- Check if the user is authenticated. If the user is authenticated then it stores the username and some other information in the session.
- ForgetPassword(string username)- Check that the username is exist or not in the system. If exist it sends the reset password link to the user's email. It also clears all the user related information from the application session.



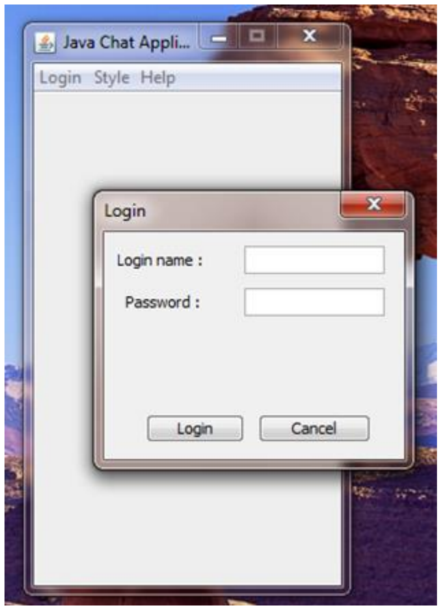
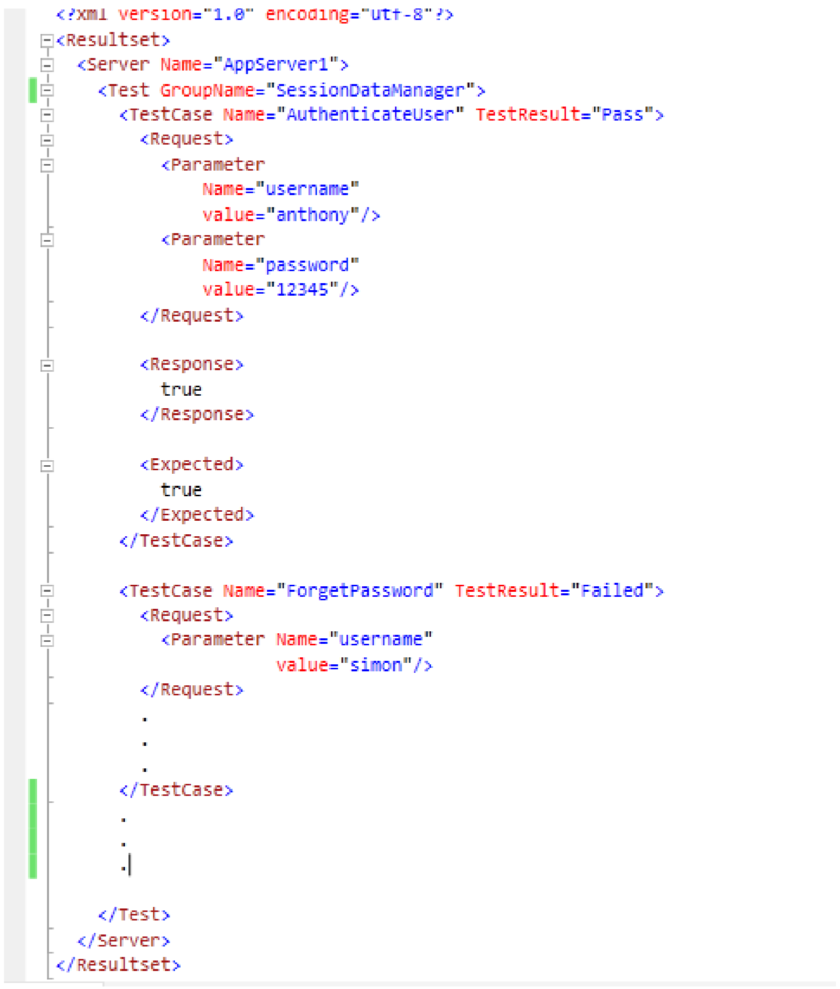


Figure 9: Class diagram of SessionDataManager class for a sample chat application

Resultset XML will have all the test cases for these four methods in a group say "Session Data Manages". Each test case has request parameters and response result. If the response result match with the expected result then the test passes, otherwise it fails. A sample resultset XML for the SessionDataManager has been given below:



```
<?xml version="1.0" encoding="utf-8"?>
<Resultset>
  <Server Name="AppServer1">
    <Test GroupName="SessionDataManager">
      <TestCase Name="AuthenticateUser" TestResult="Pass">
        <Request>
          <Parameter
              Name="username"
              value="anthony"/>
          <Parameter
              Name="password"
              value="12345"/>
        </Request>

        <Response>
          true
        </Response>

        <Expected>
          true
        </Expected>
      </TestCase>

      <TestCase Name="ForgetPassword" TestResult="Failed">
        <Request>
          <Parameter Name="username"
                     value="simon"/>
        </Request>
        .
        .
        .
      </TestCase>
      .
      .
      .

    </Test>
  </Server>
</Resultset>
```

Figure 10: Sample Resultset XML

Six classes represent the resultset xml. They are:

1. Resultset: It mainly contains the list of distributed server information using the instance of the server class.
2. Server: Server class has properties like name of the server, address of the server, binding information, contract information etc. It also contains list of the test group information.
3. Test Group: Test group contains information of one or more test cases. For example, if we consider the shopping cart module of a regular e-commerce site then we will see there are several functions under shopping cart like tax calculation, payment authorization, address verification, etc. Test group will keeps all the test result of these test cases in one group for better readability. Primarily the group formed based on the class and their properties. So, each class is a test group and the methods are the test cases under that group. But manually user can customize the group using the user interface.
4. Test Case: Test case contains the actual request and response information. Instance of request and response class is used to store the information. If any test fail then developer can get an idea that for which condition(request properties) the function doesn’t working properly.
5. Request: Contains the request information like name of the method, submitted parameters, submission method(get/post), etc.
6. Response: Response contains the result from the server against the request

Figure 11 shows the class diagram of the resultset classes.

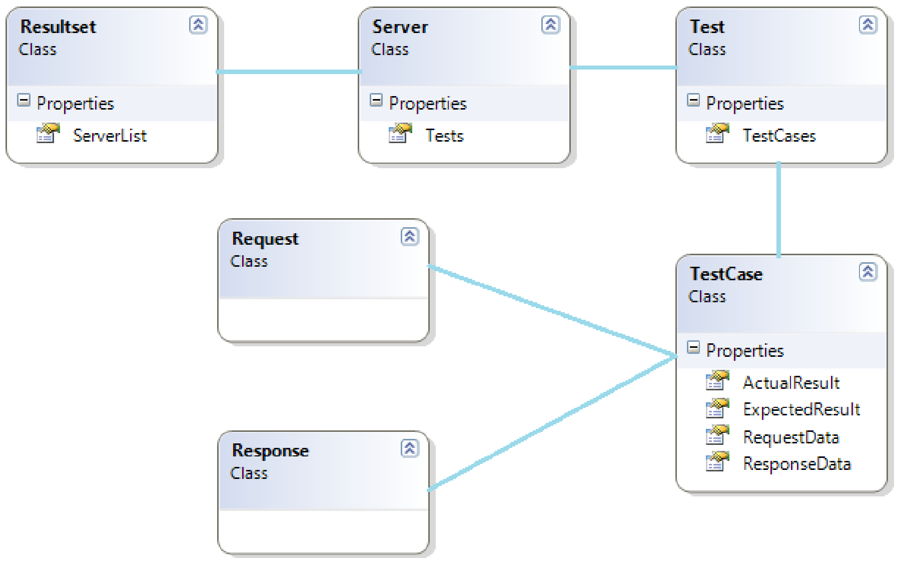


Figure 11: Resultset class hierarchies

CHAPTER 4

FRAMEWORK EVALUATION

4.1 INTRODUCTION

Automated testing framework provides benefit to a distributed system many ways. This framework can be seen as a performance tuning advisor for a distributed system. The result obtained from this framework can be used to improve the operation of the system. In this chapter we will see how we can get benefitted from this testing framework to design a successful distributed system.

4.2 PERFORMANCE TUNING ADVISOR

This is a very flexible testing framework and allows user to customize its behavior many ways. The final test result from this testing framework can be used to improve the overall distributed system. This testing framework not only explores the bugs of the system but also the behavior of the system. By seeing the test report, programmers can check which process (es)/ module(s)/ method(s) takes longer time to response than usual as it logs the testing time. They can easily find out which network or function is responsible for this particular behavior and fixed the problem. They can also find out the buggy methods very quickly by seeing the test result since the test result groups the test cases and display how many test cases fails for each test group. If most of the test cases fails for a module or takes longer time then programmers can rewrite the whole module to ensure the quality of the software. Since the framework is working by communicating multiple agents with each other runtime, it is aware about the latest changes of the system. So developer can check the whole system right after deploying an update. If the new change breaks the system partially or fully, programmer can easily notify about it and fix the issue without writing any extra test cases. So, overall this testing framework works as a performance tuning advisor for a distributed system.

4.3 TESTBED

This framework supports automated generation of mock data as well as user defined data. Also programmers can set the value of the parameters manually, if they want. Therefore, it is a very flexible framework.

4.4 Dedicated Resource

For each test case, it creates a dedicated channel using threading technique so that other test cases do not wait because of an execution of a longer test case. It also supports dependency. User can mark if a test case dependents of other test case(s). The framework will order the test cases such a way so that the parent test cases will execute before the dependent one.

4.4 SECURITY

Overall this is a very secured framework. Programmers do not need to disclose the source code of the system to the test framework through network. Since the agents are local, they read the structure of the assembly runtime and send an XML file to the

Test Engine using WCF technology. WCF ensures the security and reliability of the data.

4.5 TEST RESULT

Test result is one of the very important parts in testing. Test result designed such a way so that users can easily detect the bug of the system. Test result logs the time of each test case which helps to find inefficient functions. Test Result displays how many test cases fails in a Test Module which helps to find the buggiest areas of the system. Finally it allows the user to customize the screen so that it can be used as a report to the client.

4.6 RELIABILITY

This is a very reliable testing framework from many aspects. Most recent and proven technology has been used to build the framework. This framework supports n number of distributed server to test. It doesn’t test only middleware like other testing framework. This framework can be used to test business layer as well as the presentation and data access layer.

CHAPTER 5

CONCLUSION

5.1 CONCLUSION

An automated multi-agent test framework for distributed system has been designed and its component as well as communication process between agents has been explained briefly in this paper. It also describes how this framework can be used as unit testing tool in a distributed system. Using similar technique, this framework can be used to implement other types of testing strategies like stress testing, performance testing, load testing, etc. We believe the approach that is described in this paper can be scaled up and used to test larger and complex distributed system.

5.2 FUTURE WORK

Since, this framework is a footstep to build an automated unit testing framework for distributed system, there are several ways we can improve the logic and performance of the framework before using it in the real production environment. Currently, this framework support unit testing. In future, this framework can be extend to support other types of testing technique specially load testing, stress testing, performance testing and recovery testing which are very important for a distributed system.

REFERENCES

1. *Software Testing*. 2012 [cited 2012 August 01]; Available from: http://en.wikipedia.org/wiki/Software_testing.
2. *Functional Testing*. 2012 [cited 2012 August 10]; Available from: http://en.wikipedia.org/wiki/Functional_testing
3. *White-box testing*. 2013 [cited 2013 March 06]; Available from: http://en.wikipedia.org/wiki/White-box_testing.
4. Khan, M.E., *Different Forms of Software Testing Techniques for Finding*

Errors IJCSI International Journal of Computer Science Issues, 2010. **7**(3): p. 12.

5. Tanenbaum, A.S. and M.v. Steen, *Distributed Systems Principles and Paradigms*. Vol. 2. 2007, NJ: Pearson Prentice Hall.
6. *Introduction to Distributed System Design*. [cited 2013 March 18]; Available from: http://www.hpcs.cs.tsukuba.ac.jp/~tatebe/lecture/h23/dsys/dsd-tutorial.html.
7. *Multi agent systems*. [cited 2013 February 10]; Available from: http://www.cs.cmu.edu/~softagents/multi.html.
8. Tsai, W.T., L. Yu, and A. Saimi, *Scenario-Based Object-Oriented Test Framework for Testing Distributed Systems.* In Proc. of the Ninth IEEE Workshop on Future trends of Distributed Computing Systems, 2003: p. 288-294.
9. Ghosh, S. and A.P. Mathur, *Issues in Testing Distributed Component-Based Systems.* First ICSE Workshop on Testing Distributed Component- Based Systems, 1999.

10. Long, B. and P. Strooper, *A Case Study in Testing Distributed Systems.* in Proc. of the Third International Symposium on Distributed Objects and Applications, 2001: p. 20-29.

11. Lastovetsky, A., *Parallel Testing of Distributed Software.* ELSEVIER Trans. Information and Software Technology, 2005. **47**(10): p. 657-662.

12. Hamlet, D., *Random testing.* In Encyclopedia of Software Engineering, 1994.

13. Miller, B., et al., *Fuzz revisited: A re-examination of the reliability of UNIX utilities and services.* 1995.

14. Lyu, G.X.a.M., *Testing, reliability, and interoperability issues in the CORBA programming paradigm.* In Proceedings of the 2000 Asia-PaciJic Software Engineering Conference, 2000: p. 530-537.

15. Dowling, E.J., *Testing Distributed Ada programs*, in *Proceeding TRI-Ada '89 Proceedings of the conference on Tri-Ada '89: Ada technology in context: application, development, and deployment*1989. p. 517-527.

16. *Bindings and Security*. 2012 [cited 2013 March 10]; Available from: http://msdn.microsoft.com/en-us/library/ms731172.aspx.

VITA

Graduate School

Southern Illinois University

Ehsanul Haque

ehsan_aiub@yahoo.com

American International University Bangladesh

Bachelor of Science, Computer Science, February 2007

Special Honors and Awards:

1. Best Graduate Assistant Nominee – 2011

Southern Illinois University Carbondale, Carbondale, IL, 62901, USA

2. Employee of the quarter – 2006

Bblobal sourcing LLC, Gulshan, Dhaka, Bangladesh

www.bglobalsourcing.com

Thesis Title:

AN AUTOMATED MULTI-AGENT FRAMEWORK FOR TESTING DISTRIBUTED SYSTEM

Major Professor: Dr. Henry Hexmoor

Publications:

1. M. R. Zargham, E. Haque, and J. Cheng. 2012. KumuCloud: An Open Access Academy, 5th conference on e-learning excellence in middle east, Dubai, UAE

2. M. R. Zargham, E. Haque, and J. Cheng. 2012. KumuCloud: A Progress Report, EEE'12 - The 2012 International Conference on e-Learning, e-Business, Enterprise Information Systems, and e-Government, Las vegas, Nevada, USA