TestRig: A Platform independent system testing tool.
Candidate: Kaul Vaibhav
Advisor: Shasha Dennis

Abstract

The goal of the TestRig software is to give a test engineer a fixed interface to help him with system/integration testing of software systems. TestRig is platform independent and can be utilized to test software systems coded with any programming language. In addition to doing that, it provides templates and examples of using various Open Source testing tools to help a user design their test cases.

TestRig has been designed keeping in mind the current scenario in software development where complex systems are often created using multiple programming languages across different platforms. The challenge is to have a defined set of rules that are able to test any such system.

The software makes use of various open source testing tools to run tests and verify results, which enables a user to test a system at different levels such as Performance Testing, Blackbox Testing, and User Acceptance Testing.

TestRig is open source and utilizes a programmer's creativity to test across multiple scenarios. The thesis will show how different software systems have been tested using TestRig.

[NYU COURANT INTITUTE OF MATHEMATICS]

[TestRig]

[A Platform Independent System Testing Tool]

Vaibhav Kaul
N11167470
[Fall 2010]

A thesis submitted in partial fulfillment

Of the requirements for the degree of

Master of Science in the Department of

Computer Science, New York University

Approved: _____

(Dennis Shasha, Research Advisor)

Approved: _____

(Lakshminarayanan Subramanian, 2nd Reader)

Acknowledgements

I would like to thank Professor Dennis Shasha for his continued support and guidance. I am also grateful to Arthur Meacham, PhD candidate in CS for his help with understanding the Outsafe Project and for his inputs on testing it.

A special thanks to Professor Lakshminarayanan Subramanian for taking the time to read the thesis text and for his feedback.

Research and Development

TestRig is a platform independent System testing tool developed over Fall 2010. It started out as an application to test the Outsafe application in a cloud based platform but has been extended to incorporate other platforms as well.

Contents

Introduction

With the advent of complex software systems it has become increasingly difficult to test applications. Software companies spend millions of dollars to test their software systems and the task is getting increasingly costly due to systems getting more complex and distributed over a much wider platform base than ever before.

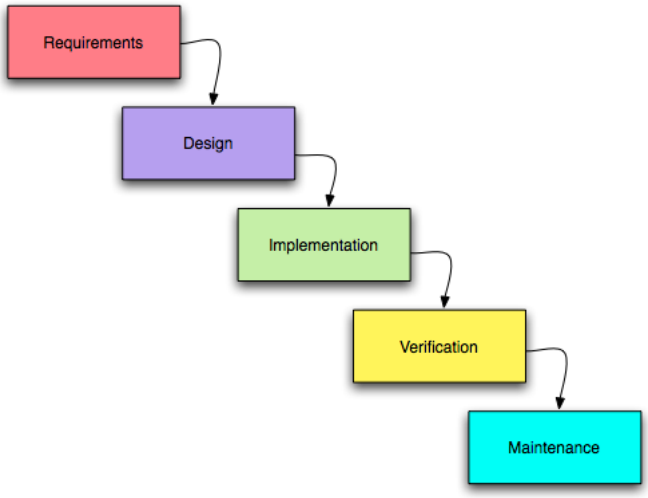
Industry is driven by rigid processes, and hence not very open to the dynamically changing software development methodologies. Nevertheless, some kind of process is the only way forward for companies employing tens of thousands of software developers. A set of interfaces is required which would act as a collection set of rules to ensure a level of completeness of Quality Assurance practices to the most complex of software as well.

TestRig is a step in that direction. It has deliberately been created in a way that makes it platform independent and simple enough for the extremely capable software development community to develop and change. At the same time, it achieves a high level of Quality Assurance.

Background

The Software Development Lifecycle

Traditional Waterfall Model



The traditional waterfall model divides the software development activities into the following parts [1]:

- **Requirement Gathering:** This is the stage where the software team tries to understand the product that needs to be developed.

- **Design:** After "Freezing" the scope of the requirement for a particular software product the software team's next objective is to come up with a good feasible design both at the high level(data flow) as well as the low level (usually code).

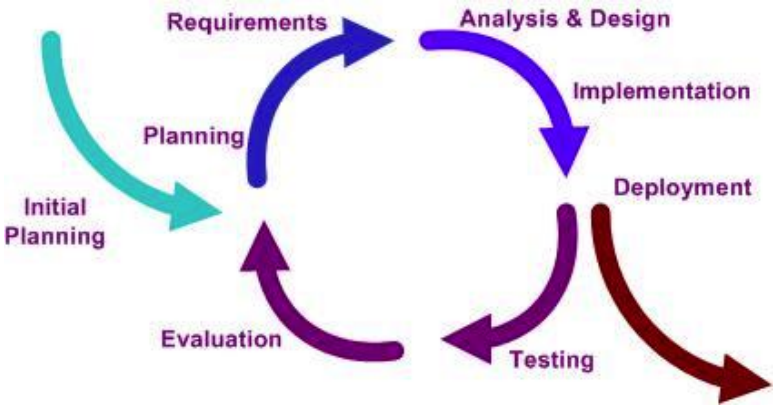
- **Implementation:** Once a design has been conceived, the team starts developing the product. The product is divided into multiple modules and members of a team or sub teams are assigned these modules for completion.

- **Verification/Testing:** Once the product has been implemented, the quality assurance activities begin. They cover a wide variety of processes, which will be discussed in much more detail in the coming text.

- **Maintenance:** This stage involves fixing anything in the product that does not meet the required client standards as well as taking care of the problems that arise due to the normal wear and tear of the system.

Iterative Development

The Classic waterfall model can no longer keep up with the rapidly changing demands of the new era of software development. The increasing demand and fickle nature of client requirements has rendered the classic method of developing software impractical. Development now does not occur in well-defined stages and one can see various stages of overlap.



Development is a continuous process and therefore it becomes crucial for testing to be dynamic too. i.e. There is no reason for “Testing” to be a well defined stage. Testing can be just as continuous as daily builds.

Software Testing

Approach

Black Box Testing

In this approach, a tester does not have access to the internal code of the application or any knowledge of the underlying data structure. The tester works with a finished product and can only test the functioning the end product.

White Box Testing

White box testing is often employed to test the completeness of the code. The tester is given access to the data structures and various algorithms and the code that implements these processes. Some examples of white box testing are API testing, Code Coverage and Fault Injection.

Types of Testing

Functional Testing

This type of testing is carried out in order to verify the agreed upon functionality of the application. Tests are carried out in parts known as Test Cases and the clients as well as the development team usually sign off on these. A software system is said to be acceptable if it passes all the test cases successfully.

Functional testing is done at multiple levels:

- Unit Testing: At the class/method level
- Integration Testing: At the module level
- System Testing: At a distributed system level

- User Acceptance Testing: Testing at the end user level
- Alpha Testing: Client gets access to software in a controlled environment
- Beta Testing: Client given access to software in his own environment

Non-Functional Testing

Non-functional testing in carried out in order to check the robustness of a software system. It is used to verify whether a piece of software can withstand invalid or incorrect inputs and still function or “fail gracefully”. It also covers an extremely important aspect known as performance testing. Some examples are:

- Performance and Load testing: The software is bombarded with high levels of data
- Stability testing: The software is run under adverse conditions such as with an overloaded processor or high RAM usage
- Security testing: Testers try and “hack” their way into a running software application

Automation Testing

As stated earlier, it is very important for testing to transform into a continuous process in the modern software development world. That said, it would be impractical and time consuming to have a team of testers manually verify software with each new build. Also, this may introduce the element of human error.

Test Driven Development (TDD)

Recent years have seen a shift towards test driven development with the advent of the xUnit, especially in their most popular forms, JUnit and NUnit. The philosophy behind TDD is that test cases be developed before the actual code is implemented. Although this might seem impractical to a seasoned programmer, it maps perfectly to client requirements. After all, the functional behavior of code is known well before the coding stage. JUnit and NUnit are very similar to each other, the latter works with the popular Microsoft .net framework whereas the former works on the Java platform. Test cases created in JUnit and NUnit are known as "Unit tests" since they typically test a unit of functionality, most commonly a function or a method through the concept of Assertions. An assertion is made during a test case where a value is compared to a test value and the unit test passes or fails depending upon which result it returns. We will see some JUnit and NUnit examples as a part of this text. It is worthwhile to know that while xUnit was created to test small parts of the program, it is a fully powerful framework within itself and is capable of anything that Java or .net are capable of.

- Sample JUnit Test

```
import junit.framework.*;
public class SimpleTest extends TestCase {
        public SimpleTest(String name)
        {super(name);}
        public void testSimpleTest() {
        int answer = 2;
        assertEquals((1+1), answer);
        }
}
```

- Sample NUnit Test

```
using System;
using NUnit.Framework;

namespace UnitTestApp.UnitTests
{
[TestFixture()]
public class Calculator_UnitTest
{
        private UnitTestApp.Calculator calc = new calc();

[SetUp()]
public void Init() {// code called before every TestCase}
[TearDown()]
publicvoid Clean()
{
    // code that will be called after each Test case
}

[Test]
publicvoid Test()
        {
            int result = calculator.Add(2, 2);

            Assertion.AssertEquals(4, result);

        }
    }// End Class
}// End namespace
```

Automation Testing Tools [2]

Some noteworthy automation testing tools are:

- HP QuickTest Professional: This is one of the costliest and most powerful automation test tool in the market today. It allows users to record/play or write complex test scenarios using VB script and descriptive programming.

- IBM Rational Functional Tester: It is a popular regression test tool.

- SilkTest: It is an excellent enterprise class testing tool.

- Selenium: It is an excellent test tool for web application testing. We will see selenium in action as a part of our examples.

- JMeter: It is an apache performance-testing tool.

- Httperf: It is a Unix command line load-testing tool.

- LoadRunner: It is an enterprise class load-testing tool used by various software giants.

Automation Testing Frameworks

Over the years, multiple automation tools have led to an important dilemma. Consider a scenario where a company called ACME had 50 automation testers writing Test Cases in HP Quick Test professional. Due to cost cutting the CTO decided to switch all automation to the open source Selenium. Now what does ACME do with all the test cases written in Quick Test Professional?

Consider another scenario: ACME wants to test an application at various data points. Test scripts written in HP or Selenium or any other popular test tool require data to be coded with the test cases. Now every time the data is changed an automation tester is required.

Yet another scenario is that the CTO at ACME chose a testing tool because he liked the reporting format. Now a new CTO has been hired. He does not like the reporting format and thinks it is not efficient. What happens if he changes the testing tool again?

The way to get around these problems is to use Testing frameworks. Frameworks act as a layer on top of automation test tools that provide integrated support for Test Script creation, data management and reporting activities in an integrated environment. Many advanced automation-testing frameworks provide the tester with a driver on which to run their test scripts. i.e. they allow the tester to write test scripts in basic

formats like XML or simply as steps in an Excel sheet and take it upon themselves to convert those steps into automation scripts.

The biggest advantage of such frameworks is the cost effectiveness of the solution. Consider the fact that a HP Quick Test Professional license can cost upwards of a few thousand dollars and a framework that allows a tester to create test scripts for QTP can be created using just open source tools [3].

TestRig is a framework. It provides the users with a layer on top of any automation tool they may need along with an operation contract for how system test cases should be written.

TestRig

Goal

Simply stated, the Goal of TestRig is to "develop an operation contract and methodology capable of meeting test requirements for various forms of testing including but not limited to Unit Testing, System/Integration Testing and performance testing".

While this is not an easy task, a valiant attempt has been made to try to develop such a contract whose completeness is measured by the spectrum of test case scenarios it is capable of allowing. Its practicality has been demonstrated in a distributed application running on a cloud developed by Arthur Meacham and Dennis Shasha at NYU. I will show TestRig in action for programs as simple as a Lab Assignment to a fully distributed outsourced database management system. We will also cover various levels of testing for applications having no user interface like web services to applications with a rich user interface like Facebook. Performance Test tools like JMeter will also be "TestRigged" to show its completeness.

State of the Art

TestRig is implemented in Java. The Java platform was chosen to allow the interface to be used on multiple platforms. This however does not imply that TestRig can be used only to test Java programs. As will be shown later, TestRig can test software written with any language on any platform.

The basis of TestRig is a powerful operation contract.

The TestRig Operation Contract

The TestRig operation contract is merely an interface that needs to be followed in order to test an application. The operation contract acts as a pact between the software and the tester that ensures that the software in question is fully tested. It is expected that each tester will either code an implementation of this interface or use the existing examples for TestRigging.

The interface consists of various functions that I believe will cover all activities that are essential to test software. The reason this has been exposed as an Interface (as opposed to an Abstract Class) is that the tester is forced to implement all functions explicitly. It should however be noted that there is nothing in place that would stop a tester from creating an adapter layer in the middle and simply ignoring some of the functions which are meant to be overridden. It would however be advisable to take this path only if a tester is absolutely certain that he does not need to bother with the method in question.

The Java Interface has functions for the following:

- **Initialize System:** This function is meant to be overridden with a method that performs all activities related to initializing the system. This could mean different things for different types of software. For a simple Lab assignment this function

could be used to check if all prerequisite libraries are present on the machine and for a large scale application like Outsafe this could mean setting up and starting machines in the Cloud.

- **Move Files into place**: This function will be overridden by code that will ensure all the files have been moved into place before the actual test operation can start. It is essential that we do not take this to mean that we are merely copying files from one directory to another; rather this could be used to send files over various networks or platforms using protocols such as FTP, SFTP or SCP.

- **Execute Process**: This function will be integral to testing. This will be used to execute scripts on a local or a remote machine. This process can be made especially useful for applications that support command line options. Most software systems coded in either Java or.net have that facility. The Upcoming Outsafe example makes use of this to call scripts on remote UNIX machines in the Cloud to accomplish the task of testing a distributed software system very effectively.

- **Wait:** This method may be utilized for performing operations while we wait for scripts to execute. This may not be appropriate for all scenarios but may be used in scenarios that require long testing runs.

- **Get Results:** This method will be utilized for getting results into place after a test run for analysis. In most cases the Move files method may be used in its place.

- **Parse Results:** This method will be used to parse or make sense of any retrieved results from output folders on local machines or from the Cloud.

- **Assert Conclusion:** This method should be called right after parsing the results and should utilize either the pass and fail methods from TestRig or if JUnit is being utilized should use its methods to pass or fail a test.

- **Pass Test:** This method should contain code that explicitly passes a test case. In the simplest of terms it should contain some last minute checks and then simply call the JUnit pass method.

- **Fail Test:** Similar to pass test, it should perform some book keeping if required and call the JUnit fail method.

- **Cleanup:** This method should be called at the end of the system test. This could be as simple as removing files from directories to the more complex task of shutting down all machines in the cloud.

It is quite understandable that all of these methods may not apply to all forms of testing. In such a case the tester should implement the Operation Contract using an Adapter class and finally extend that class in the actual system test case.

The purpose of this operation contract is to maintain consistency in system testing practices. Therefore it is essential that a tester stay as close as possible to the contract. This will ensure architectural consistency in test cases and will make them more portable. It will establish a commonality between different test cases and will allow other testers to easily maintain or modify test cases created by a tester.

What sets it apart

What clearly sets TestRig apart is the fact that instead of going after different technologies that provide assertion capabilities in test cases, TestRig is an attempt to formalize an operation

contract. It is meant to use existing testing tools to implement the system test cases at a lower level but it wraps the system test using an interface which ensures structural consistency across different types of test cases. This is essential for software teams growing at a fast rate. With the sheer number of people working in collaboration that we see today it is essential that this contract be established to ensure that system testing knowledge is not esoteric.

The TestRig operation contract has been implemented in Java to ensure portability across different platforms. It must be stated that although it is advised to use the operation contract as it is defined, there is no harm in porting it over to a different language. The programming language is merely a tool, which has been used to implement the idea of structural consistency. If a tester feels that portability across platforms is not an issue it would do no harm to port the contract over to a language like C# or VB (preferably C# since it is much closer to Java).

TestRig is a first attempt to reach some common ground between different types of system testing carried out in an organization. It is open source and hence is open for modifications as long as the concept of structural consistency is kept intact.

In addition to the Operation contract it also aims at giving the user examples or templates using various open source-testing tools which will be useful for them in designing their test cases.

In the following section we will see how TestRig incorporates the operation contract across a wide variety of testing scenarios. The list is certainly not exhaustive but is extremely diverse. As this list gets longer, TestRig will evolve and try to incorporate many more features that will allow a wider variety of Test scenarios to be TestRigged.

Examples

In the following example we will look at different scenarios that are generally tested while testing s system. For each example will go through the context of the problem and the tools that are used to create and test that piece of software. For certain specialized tools like JMeter and Selenium the context will include a short tutorial that is meant to serve as a template in itself.

In the operation contract implementation we will look at the methods that need to be over ridden from the Operation contract interface. We will find the methods mentioned in the contract are sufficient to test this wide variety of scenarios.

The above-mentioned points emphasize the goal of TestRig. They take care of providing the testers with a template for testing these scenarios as well as stay true to the operation contract interface.

Test whether a number is prime or not

Context

What we are essentially testing for is primarily of a number. Although this is not a standard system in testing scenarios, it is being used merely to demonstrate how the operation contract can be used to test simple lab assignments.

Tools Used

This example is coded using Java and will not be making use of any testing frameworks like JUnit or TestNG. We will simply be running the program with command line options. For demonstration purposes, we will write the result to a text file on the file system, which will be picked up and parsed by TestRig. Assertions will be made by TestRig.

We will assume a simple Java program called Prime.Java which contains a main method which writes to an output file if the number entered on the command line is prime or non prime.

Operation Contract Implementation

The operation contract implementation for this example will use the following methods from the interface.

- We will use the **moveFile** method to make sure the java file is in a directory that is on the classpath.

- Once the files have been moved over we need to run the java program. We do this in the **executeProcess** method.

```
String complile="javac  Prime.java";

String execute=”java Prime 7”;

Runtime runtime=Runtime.getRuntime();
        String output="";
        try {
                Process p=runtime.exec(compile);
                String line="";
                BufferedReader outCommand = new BufferedReader(new
                                InputStreamReader(p.getInputStream()));
                while((line=outCommand.readLine())!=null)
                {
                        output+="<br>"+line;
                }
                p=runtime.exec(execute);
        } catch (IOException e) {
                // TODO Auto-generated catch block
                return e.toString();
        }
```

- At this time all we need to do is use the **parseResults** to read the output file and in the **assertConclusion** method we need to pass or fail the test using **passTest** or **failTest**.

JUnit Test Case

Context

The JUnit test case has simply been designed to test the working of a simple calculator. We will be testing the usual operation along with the boundary conditions.

Tools Used

Much like the earlier one, the present example uses simple Java along with the JUnit library to test the different functions of a calculator. The only difference in this case is that the results are not written to an output file so we don't need to move files around or parse them. We will simply be looking at how JUnit can be used with TestRig. Java allows us to execute JUnit test cases from Java Code. We will use that functionality to invoke JUnit test cases that we have designed to run the test cases.

We will write the test cases to test the "add", "subtract", "multiply" and "divide" methods in the CalculatorTest.java class.

Operation Contract Implementation

- We will use the **moveFile** method to make sure the java file is in a directory that is on the classpath.

- Once the files have been moved over, we need to run the JUnit test. We do this in the **executeProcess** method:

```
String complile="javac  CalculatorTest.java";

Runtime runtime=Runtime.getRuntime();
                String output="";
                try {
                        Process p=runtime.exec(compile);
                        String line="";
                        BufferedReader outCommand = new BufferedReader(new
                                        InputStreamReader(p.getInputStream()));

                       JUnitCore.runClasses(CalculatorTest);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        return e.toString();
                }
```

- Since this is using JUnit internally, it will handle the assertions automatically so one does not need to over-ride those methods.

UI Testing in Windows (C#)

Context

In this example we will be testing the UI functionality. As a test application, we have chosen Facebook to demonstrate what can be achieved using TestRig. The scenario will be extremely simple since we will simply be testing the successful sending of a Facebook message. Again this example will not be writing any output files or parsing any results. Since we are using the Selenium testing tool, it will fail the test case automatically if a particular control is not found or is misaligned.

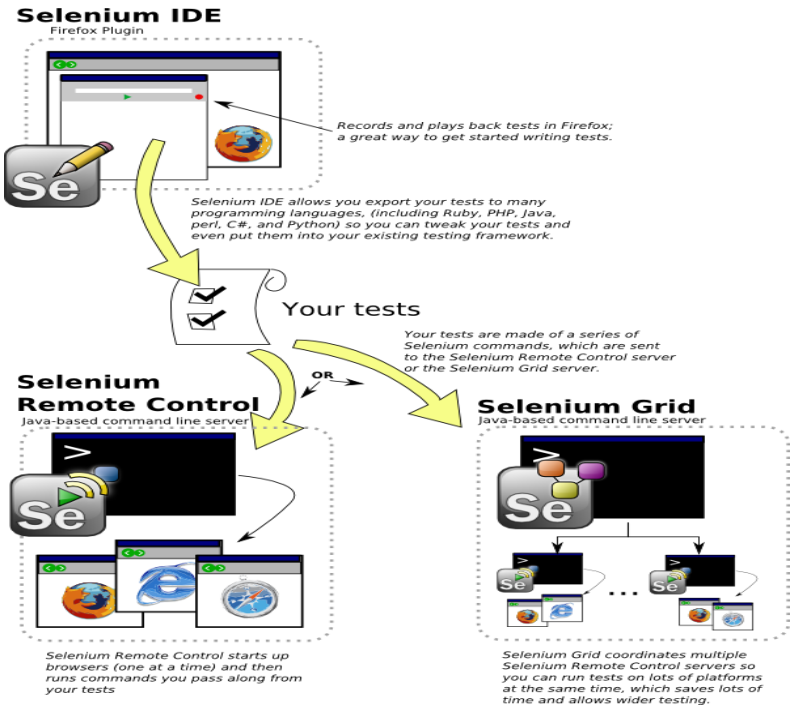
Tools Used

For this example we will be using an Automation Testing tool called Selenium.

Selenium

Selenium is a set of tools specifically designed for web application testing.

How it Works



Selenium is available for download as a Firefox add-on. This small download enables us to start "Recording" our test cases. What it means by recording is that once we enable Selenium we can start performing action on a website. Everything that we do is recorded as steps in a test case. Each step will basically have a web control like a text box or a button along with an action like Set-Text or Click associated with it. Certain steps like setting text in a Text Box will also have values associated with them.

While this may be enough for testing basic websites, it can be very difficult to test dynamic websites making use of java scripts and AJAX. This is because these websites create web controls like text boxes or buttons dynamically. There may be a possibility that while "recording" our test case the web control that we want to test does not even exist. If this is the case we do not know the identifier for a particular web control, which makes it very difficult to work with it.

This is where the Selenium IDE would prove to be very helpful. The Selenium IDE enables us to convert the native selenium test steps into a host of different programming languages like C# (NUnit), Java (JUnit), Python etc. This combined with the selenium API can be an extremely powerful tool as will be displayed in the example.

Even after being able to do all this, there are major challenges a UI tester faces. One of the biggest ones is the "event-loop". What this means is that in modern websites with AJAX and java script, things are not always what they seem to be. For instance, in an address bar on a web application while we start typing an address, the AJAX kicks in and gives us the auto-complete feature. Hence, a simple call to Set-Text in such a field will fail. Ajax internally uses events like "Key-Press" or "Key-Down" to fire itself. These events are not invoked when an automation tool like Selenium calls "Set-Text". For this reason, Selenium introduced an API call named KeyPressNative. What this function does is, that instead of setting text in a text box, it mimics key presses by a user. This poses yet another challenge as far as

automation testing goes. Since the API now is is setting native text only, we can no longer rely on it to set this text in a particular control i.e. we have the following two functions in question:

1) Type(String ControlName, Sring Value) : This sets Value in the Control identified by control name

2) KeyPressNative(String key) : Simply mimics the user pressing a key

Hence we can quite clearly see that it is now the Testers responsibility to ensure that proper control is "In-Focus" when we use the KeyPressNative API call.

We will illustrate an example of this when we send a Facebook message through Selenium while typing the address we want to send the message to.

Another point worth noting is the management of web control identifiers. While it is possible to simply insert the web control names in the C# code itself, what happens when the identifier for the control in question changes? For this purpose we will make use of the concept of an Object Repository borrowed from the Automation Tool Quick Test Professions.

What an Object repository essentially does is that it maps a control name to a control identifier. The control name is significant only as far as the TestCase is concerned while the identifier is actually what identifies it on a web page. In quick test professional, this is done through a specialized file known as in Object Repository but for our example we will use a simple XML file to map control names to their identifiers on the page. Something similar will be done for the user accounts that we will be using.

For e.g.

```
<control>
        <id>txtBoxUserName</id>
        <name>email</name>
        <type>Text Box</type>
</control>
```

This signifies a single Facebook control. The id filed contains the identifier on the actual webpage and name filed is what identifies this control in out test case. If Facebook decides to change the identifier for the text box where users enter their usernames while logging in, we can simply edit the XML file and the test case will still run fine.

For users :

```
<user>
        <id>user1</id>
        <name>ERICH YATE</name>
        <email>erich.yate@gmail.com</email>
        <password>erich_yate</password>
</user>
```

This is a single user as stored in the users XML file. This allows us to relate test cases to usernames by simply using their ids. When user accounts expire, all we do is create new accounts and edit these XML files to continue running our test cases.

In conclusion, it is far easier to change XML files than to change actual code since it won't require the testing libraries to be rebuilt. I would strongly suggest this be done when using test rig to test UI functionality.

Another Selenium feature is how it actually identifies the webpage elements. Currently it supports the following methods [5]:

identifier=*id*

Select the element with the specified @id attribute. If no match is found, select the first element whose @name attribute is *id*. (This is normally the default; see below.)

id=*id*

Select the element with the specified @id attribute.

name=*name*

Select the first element with the specified @name attribute.

- username
- name=username

The name may optionally be followed by one or more *element-filters*, separated from the name by whitespace. If the *filterType* is not specified, **value** is assumed.

- name=flavour value=chocolate

dom=*javascriptExpression*

Find an element by evaluating the specified string. This allows you to traverse the HTML Document Object Model using JavaScript. Note that you must not return a value in this string; simply make it the last expression in the block.

- dom=document.forms['myForm'].myDropdown
- dom=document.images[56]
- dom=function foo() { return document.links[1]; }; foo();

xpath=*xpathExpression*

Locate an element using an XPath expression.

- xpath=//img[@alt='The image alt text']
- xpath=//table[@id='table1']//tr[4]/td[2]

link=*textPattern*

Select the link (anchor) element which contains text matching the specified *pattern*.

- link=The link text

css=*cssSelectorSyntax*

Select the element using css selectors.

Sending the Facebook Message

Now we look at the code used to send the actual Facebook message. In true TestRig spirit even the TestCase will make use of an operation contract. This has been done since many social media websites allow a user to perform similar action. The reason an operation contract has been added is to allow a web tester to write a new implementation and switch the Facebook implementation.

In this case I have used C# to test the web-application simply to demonstrate the wide reach of Test Rig. I was able to accomplish this in Java as well using Selenium and JUnit. The C# test case was ported from that implementation since C# is very similar to Java and NUnit is simply JUnit ported to C#.

Let us look at the Base Class for our Selenium Tests:

```
public class SeleniumHelper
  {
    /// <summary>
    /// The Selenium Object
```

```
/// </summary>
public ISelenium selenium;
public Dictionary<String, User> users = new Dictionary<String, User>();
public Dictionary<String, Control> controls = new Dictionary<String, Control>();
public SeleniumHelper(ISelenium selenium, String controlsfileName, String usersFileName)
{
   this.selenium = selenium;
   users = Configuration.loadUsersFromXML(usersFileName);
   controls = Configuration.loadControlsFromXML(controlsfileName);
}
/// <summary>
/// This function logs a user into the website using a
/// specified User name and password identifier from the
/// property file.

/// </summary>
/// <param name="user"></param>
public virtual void login(String user)
{
}
/// <summary>
/// This function Log's out of the current session.
/// </summary>
public virtual void logOut()
{
```

```
        }
        /// <summary>
        /// This function clicks a specified link name.
        /// </summary>
        /// <param name="linkName"></param>
        public virtual void clickLink(String linkName)
        {
        }

        /// <summary>
        /// This function clicks a specified button.
        /// </summary>
        /// <param name="buttonName"></param>
        public virtual void clickButton(String buttonName)
    { }
/// <summary>
/// This function sets text to a given text area/ text box.
/// </summary>
/// <param name="controlId"></param>
/// <param name="text"></param>
public virtual void setText(String controlId, String text)
{
}
/// <summary>
/// This functions sends actual keystrokes to the given screen
```

/// irrespective of what control has been selected.

/// </summary>

/// <param name="text"></param>

public virtual void setNativeText(String text)

{

}

/// <summary>

/// This function verifies the presence of Text on a page.

/// </summary>

public virtual void verifyTextPresent()

{

}

/// <summary>

/// This function is used to send a message

/// </summary>

/// <param name="to"></param>

/// <param name="subject"></param>

/// <param name="msgTxt"></param>

public virtual void sendMessage(String to, String subject, String msgTxt)

{

}

/// <summary>

/// This function is used to send multiple messages.

/// </summary>

/// <param name="to"></param>

```
/// <param name="subject"></param>
/// <param name="msgTxt"></param>
public virtual void sendMultipleMessages(List<String> to, List<String> subject, List<String> msgTxt)
{
}
/// <summary>
/// This function will return the href url for the given link name
/// </summary>
/// <param name="linkName"></param>
/// <returns></returns>
public virtual String getUrlFor(String linkName)
{
return "";
}

/// <summary>
/// This function will return all the links in the form of a Hashtable on the page
/// </summary>
/// <returns></returns>
public virtual Hashtable getAllLinks()
{
return null;
 }
/// <summary>
/// Sets the Test Case Execution Speed
```

```
/// </summary>
/// <param name="delay"></param>
public virtual void setSpeed(long delay)
{
selenium.SetSpeed("" + delay);
}
}
```

Some of the important functions in the implementation are:

1) **private String getControl(String name)**

   This retrieves the control identifier from the XML file by looking at its name.

2) **public SeleniumHelper(ISelenium selenium, String controlsfileName, String usersFileName)**

   This serves as the constructor of the Selenium implementation. The parameters include the Selenium object instantiated in the TestCase. This selenium object is used to call the selenium API. Along with this object we also pass in the object repository file name and the users XML file.

3) **public virtual void login(String userID)**

   This function will allow a user to login using a username and password picked up from the users XML file using the id provided.

   Implementation for Facebook:

```
public override void login(String user)
{
print("Logging into Facebook with Username:"+getUser(user).email+" password:"+getUser(user).password);
//Open the URL
selenium.Open(getControl("startUrl"));
```

```
//Wait for page to load
waitForPage();
//Type in the URL
selenium.Type(getControl("txtBoxUserName"), getUser(user).email);
//Type in the password
selenium.Type(getControl("txtBoxPass"), getUser(user).password);
//Click Login Button
selenium.Click(getControl("btnLogin"));
waitForPage();
}
```

4) **private User getUser(String id)**

This function returns a User object from the XML file by looking at its id.

5) **public override void setNativeText(String text)**

This function wraps the "KepPressNative" selenium API call by calling the method for each character of the given string.

Implementation for Facebook:

```
public override void setNativeText(String text)
    {
      for (int x = 0; x != text.Length; x++){
String ss = "" + (int)text[x];
selenium.KeyPressNative(ss);
Thread.Sleep(200);
 }
}
```

6) **public override void sendMessage(string to, string subject, string msgTxt)**

This function is used to send the actual message in Facebook. This is by far the most challenging of all function implementations. One of the reasons is that we need to get the name of the actual message text box by parsing the page HTML since the text box is dynamically created so we cannot store its name in advance.

The other challenge is using the event-loop as was explained earlier.

Implementation for Facebook:

```
public override void sendMessage(string to, string subject, string msgTxt)
    {
      waitForPage();
      selenium.Click(getControl("linkMessages"));
      waitForPage();
      Thread.Sleep(3000);
      selenium.Click("link=" + getControl("linkNewMessage"));
      Thread.Sleep(3000);
      selenium.SetSpeed("2000");
      String html = selenium.GetHtmlSource();
      char[] sep = { '\n' };
      String[] lines = html.Split(sep);
      String subline = "";
      foreach (String line in lines)
      {
        if (line.IndexOf("=\"ids_") > 0)
        {
```

```
                subline = line.Substring(line.IndexOf("=\"ids_"), line.IndexOf("=\"ids_") + 50);
                subline = subline.Substring(2);
                subline = subline.Substring(0, subline.IndexOf("\""));
              }
            }
            selenium.Focus("xpath=//*[@id=\"" + subline + "\"]");
            Thread.Sleep(1000);
            selenium.KeyPressNative("9");
            Thread.Sleep(1000);
            setNativeText(getUser(to).name);
            selenium.Type("name=" + getControl("txtSubject"),subject);
            selenium.Type("name=" + getControl("txtMsgBody"), msgTxt);
            selenium.Click(getControl("btnSend"));
          }
```

Once the Operation contract has been implemented all we need is the actual test case. In our case we will be using the following test case:

```
[TestFixture]
 public class Test
{
private ISelenium selenium;
private StringBuilder verificationErrors;
 [SetUp]
public void SetupTest()
      {
         selenium = new DefaultSelenium("localhost", 4444, "*chrome", "http://www.facebook.com/");
```

```
        selenium.Start();
        verificationErrors = new StringBuilder() }
 [TearDown]
public void TeardownTest()
{
try
{
          selenium.Stop();
}
catch (Exception)
{
}
}
 [Test]
public void SendMessage()
{
Facebook facebook = new Facebook(selenium, "facebook.xml", "FBusers.xml");
facebook.login("user1");
String random = RandomStringGenerator.GetRandomString();
facebook.sendMessage("user1", "hi", random);
facebook.logOut();
Assert.IsTrue(true);
}
```

The last line will pass the test. The reason there are no explicit failures is because if Selenium does not find a particular control it will automatically fail the test case. So we do not need to handle assertions explicitly.

Performance Test For WCF Web Service

Context

The WCF Web Service that is being tested is a real world project undertaken by the author at CommonDesk, LLC. It exposes the functionality of the extremely popular archiving tool, Symantec Enterprise Vault, in the form of a web-service. It uses REST architecture and returns JSON objects. This enables the service to be used by multiple platforms like Java, XCode, Blackberry JDE, Android and Windows Phone 7 among others.

After completion of coding, the author was tasked with the process of benchmarking/improving the performance for the web-service. After looking at various open source tools it was decided that Apache JMeter would be used for this purpose. This example will show us how JMeter can be used to test performance of a system and also, we will look at how it can be Test Rigged using the service contract.

Tools Used

Microsoft WCF [6]

The Windows Communication Foundation (or WCF) is an application-programming interface (API) in the .NET Framework for building connected, service-oriented applications.

A WCF client connects to a WCF service via an Endpoint. Each service exposes its contract via one or more endpoints. An endpoint has an address, which is a URL specifying where the endpoint can be accessed and binding properties that specify how the data will be transferred.

The mnemonic "ABC" can be used to remember Address / Binding / Contract. Binding specifies what communication protocols are used to access the service, whether security mechanisms are to be used, and the like. WCF includes predefined bindings for most common communication protocols such as SOAP over HTTP, SOAP over TCP, and SOAP over Message Queues, etc. Interaction between WCF endpoint and client is done using a SOAP envelope. SOAP envelopes are in simple XML form that makes WCF platform independent.

When a client wants to access the service via an endpoint, it not only needs to know the contract, but it also has to adhere to the binding specified by the endpoint. Thus, both client and server must have compatible endpoints.

With the release of the .NET Framework 3.5 in November 2007, Microsoft released an encoder that added support for the JSON serialization format to WCF. This allows WCF service endpoints to service requests from AJAX-powered web pages.

The WCF Interface

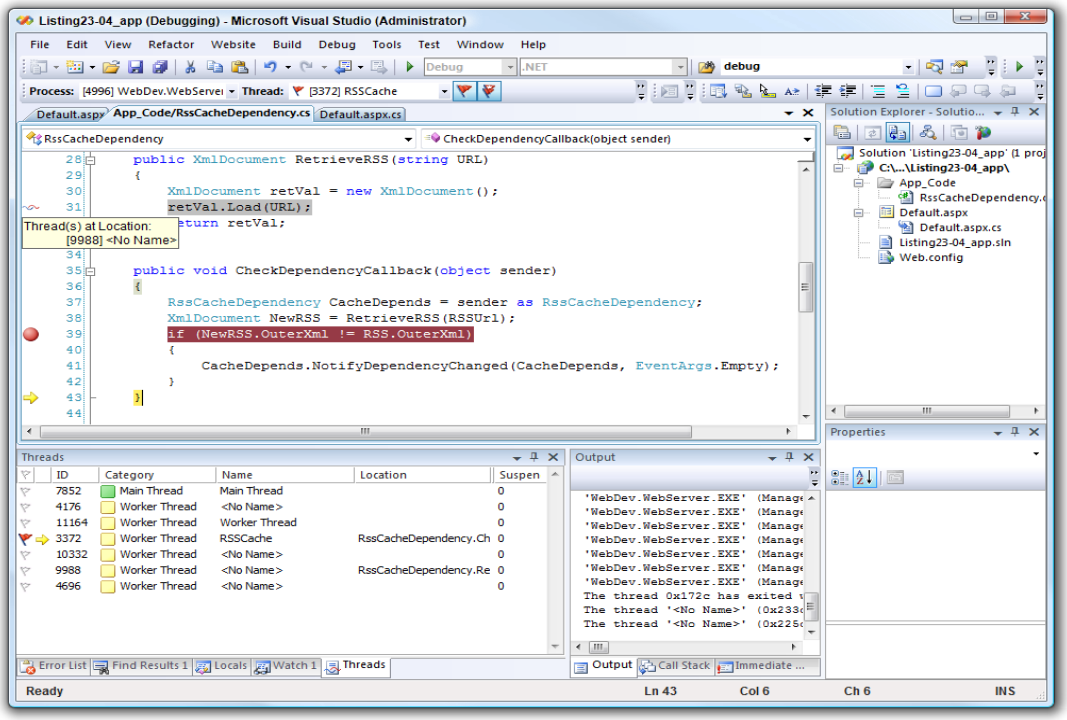
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using cd_ev_service.objects;

namespace cd_ev_service
{
   [ServiceContract]
```

```
public interface Icd_ev_service
{
    [OperationContract]
    [WebGet(UriTemplate = "/Login/{username}/{password}/{domain}", ResponseFormat =
WebMessageFormat.Json)]
    String Login(String username, String password, String domain);
    [OperationContract]
    [WebGet(UriTemplate = "/Logout/{token}", ResponseFormat = WebMessageFormat.Json)]
    String Logout(String token);
    [OperationContract]
    [WebGet(UriTemplate = "/GetMailFolders", ResponseFormat = WebMessageFormat.Json)]
    EV_FolderItem GetMailFolders();



    [OperationContract]
    [WebGet(UriTemplate = "/GetItemsInFolder/{folderName}", ResponseFormat = WebMessageFormat.Json)]
    List<EV_MessageItem> GetItemsInFolder(string folderName);

   [OperationContract]
    [WebGet(UriTemplate = "/GetFullItem/{ssid}/{archiveID}", ResponseFormat = WebMessageFormat.Json)]
    EV_MessageItem GetFullItem(string ssid, string archiveID);

    [OperationContract]
    [WebGet(UriTemplate = "/SimpleSearch/{term}/{vaultName}", ResponseFormat = WebMessageFormat.Json)]
    List<EV_MessageItem> SimpleSearch(String term, String vaultName);

    [OperationContract]a
    [WebGet(UriTemplate = "/AdvancedSearch/{queryString}", ResponseFormat = WebMessageFormat.Json)]
    List<EV_MessageItem> AdvancedSearch(String queryString);
```

```
    [OperationContract]

    [WebGet(UriTemplate = "/DeArchiveItem/{itemID}/{archiveID}", ResponseFormat =
WebMessageFormat.Json)]

    bool DeArchiveItem(String itemID,String archiveID);

    [OperationContract]

    [WebGet(UriTemplate = "/GetArchives", ResponseFormat = WebMessageFormat.Json)]

    List<EV_Archive> GetArchives();

  }

}
```

Visual Studio

The Microsoft visual studio is simply an advanced IDE for creating .net based projects. When a programmer uses VS he has the option of starting with a set template (WCF service in this case) that basically takes care of configuration properties. The visual studio environment also provides a user with the debugging tools that can be used to debug programs locally or remotely by attaching to processes.



Apache JMeter

"Apache JMeter may be used to test performance both on static and dynamic resources (files, Servlets, Perl scripts, Java Objects, Data Bases and Queries, FTP Servers and more). It can be used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types. You can use it to make a graphical analysis of performance or to test your server/script/object behavior under heavy concurrent load". [7]

JMeter Features

We can use JMeter to test the following types of applications:

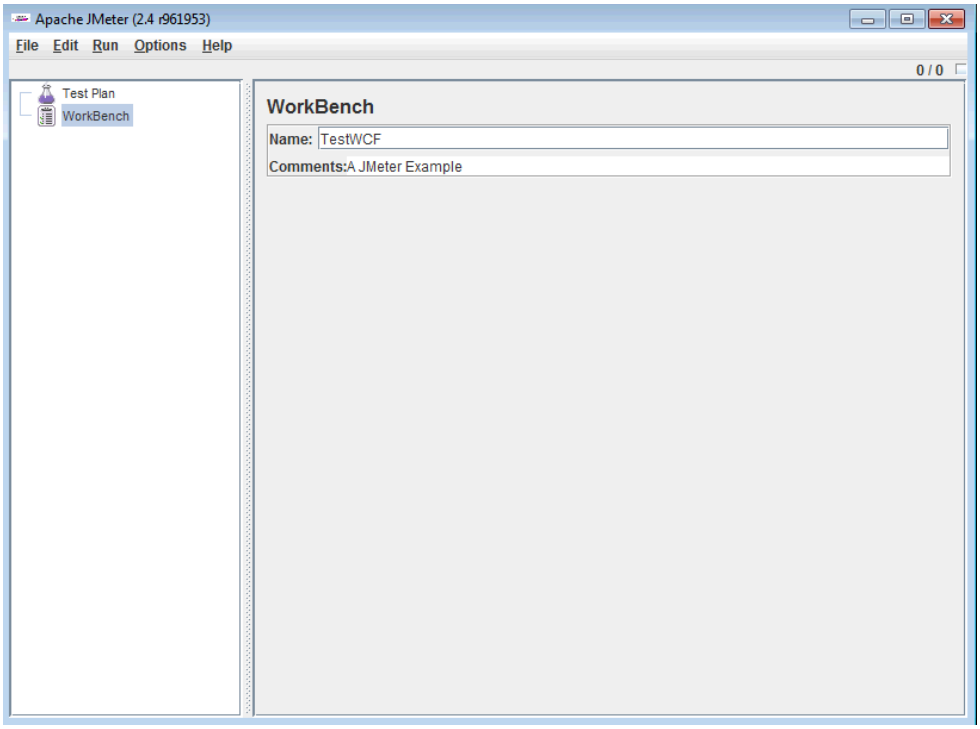
- Web - HTTP, HTTPS
- SOAP
- Database via JDBC
- LDAP
- JMS
- Mail - POP3 and IMAP

Creating a Simple JMeter Test

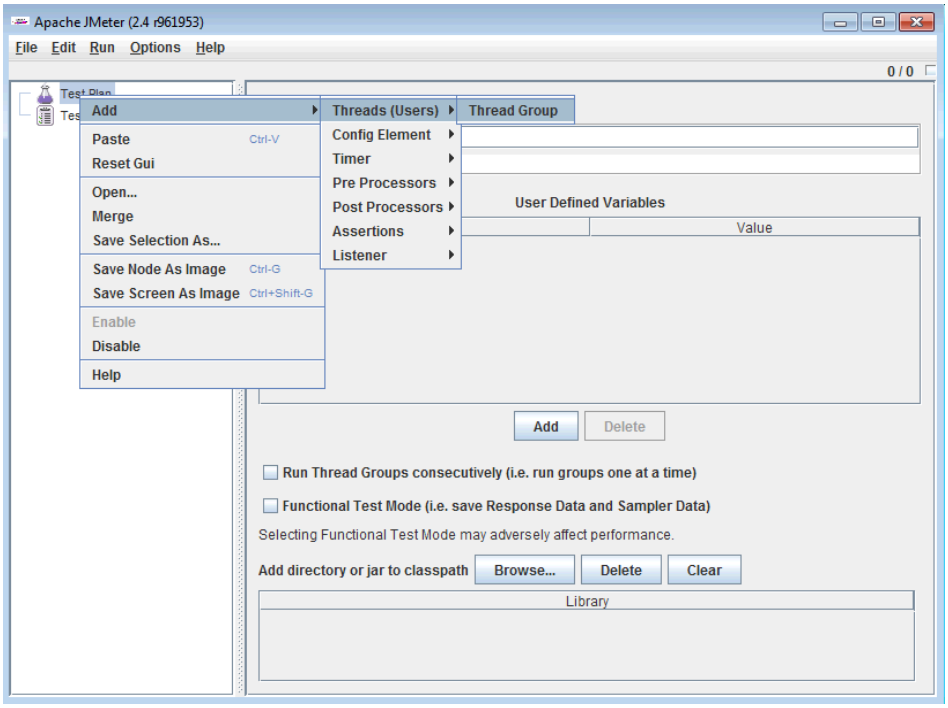
We can download the latest version from the Apache website. Once downloaded, one can simply navigate to the bin folder and run jmeter.bat (windows) or ./jmeter (unix).

We will create a JMeter test that will simulate the load of 100 simultaneous users calling the web-service from their client devices.

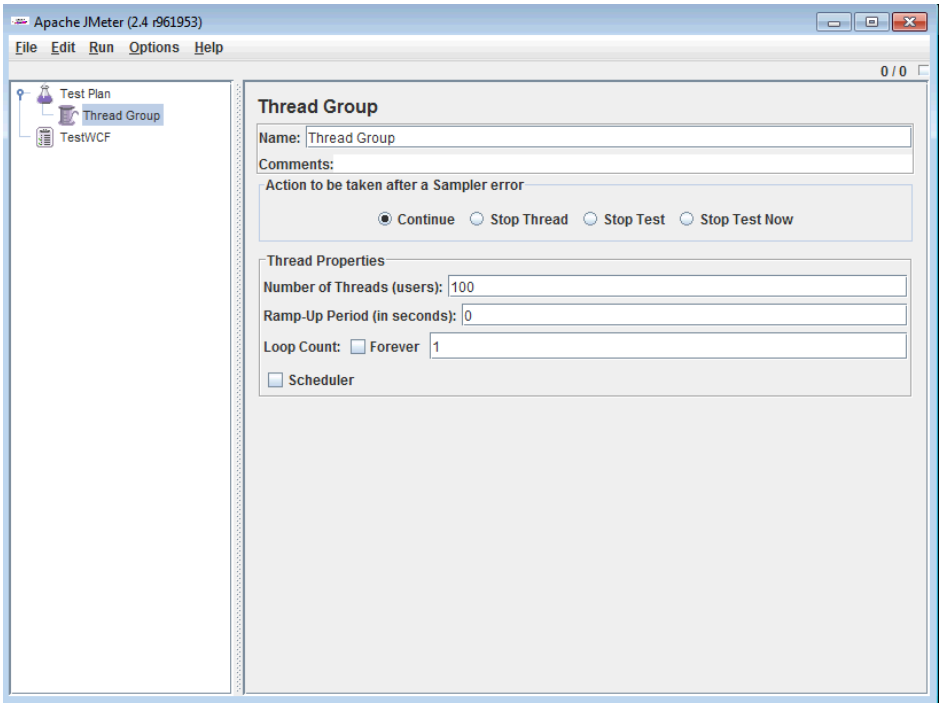
Step 1 : Open JMeter GUI



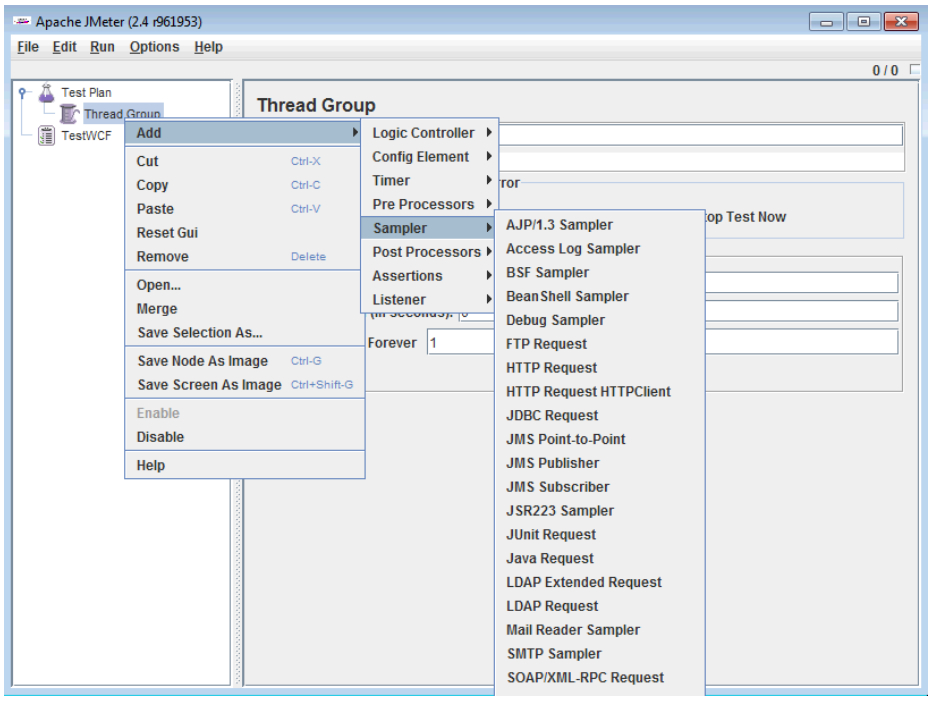
Step 2 : Add a Thread Group. This serves as the number of users we want to simulate.



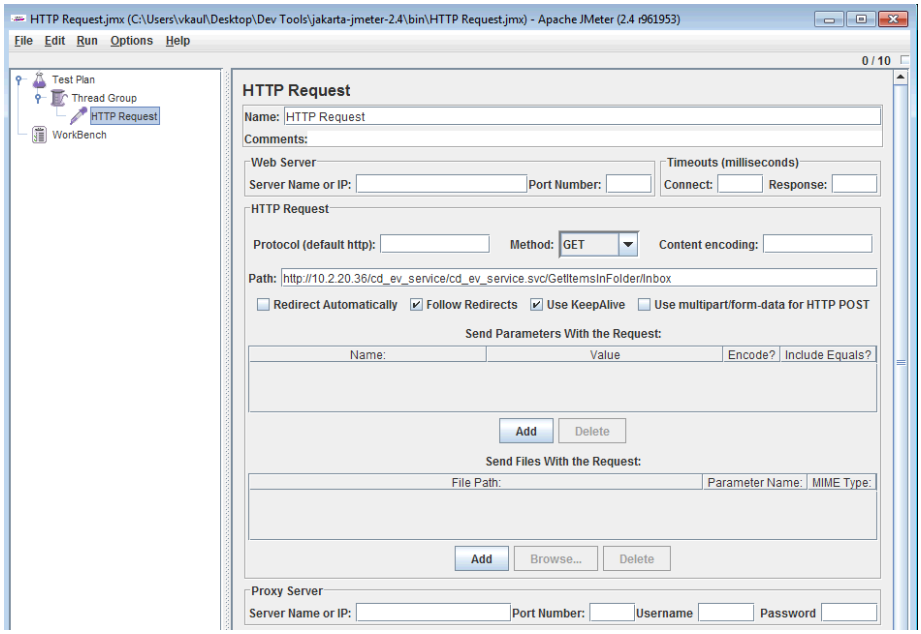
Now Enter 100 as the number of users with a ramp up period of 0 to simulate simultaneous connections.



Step 3: Add HTTP Sampler

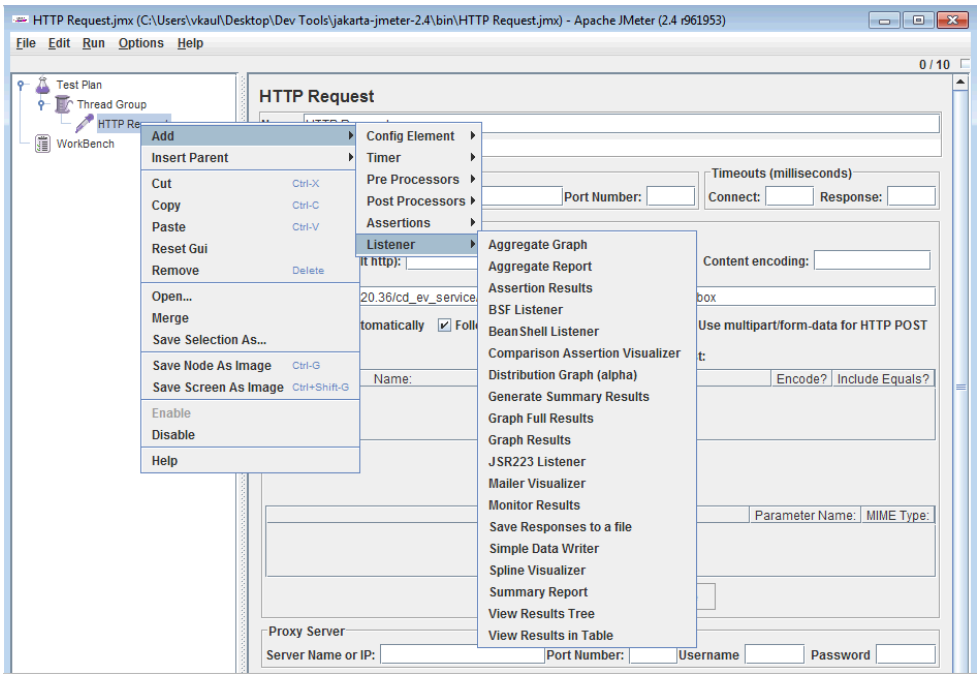


In the path write the URL of the service method that you would like to test.

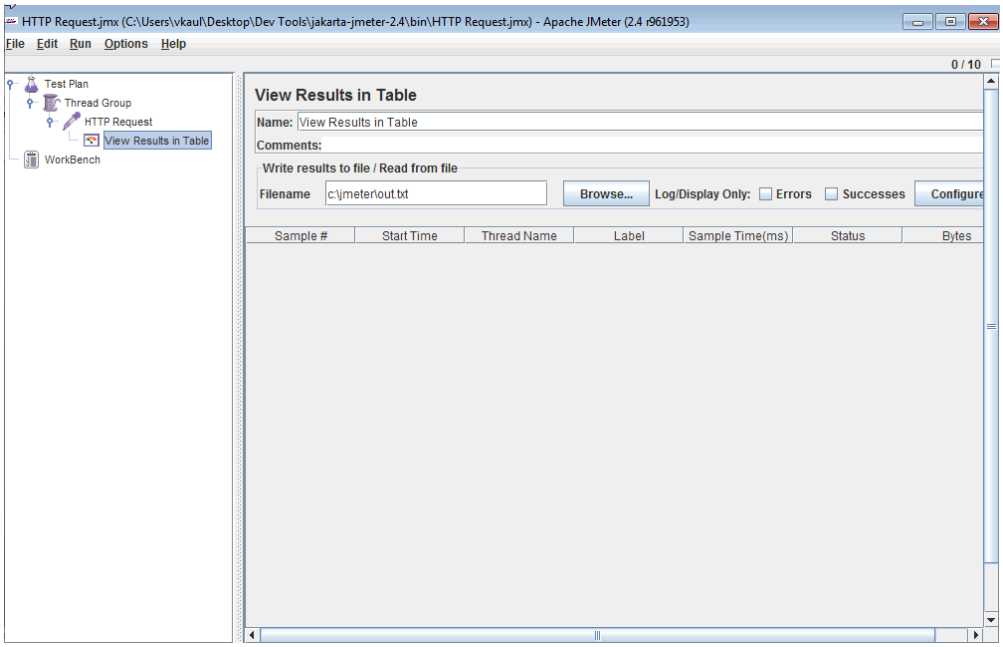


At this point we have done enough configurations to start sending the requests. But in order to view results we need to add a listener. For our example we will use a simple table to view results and write them to a file.

Step 4: Adding the Listeners

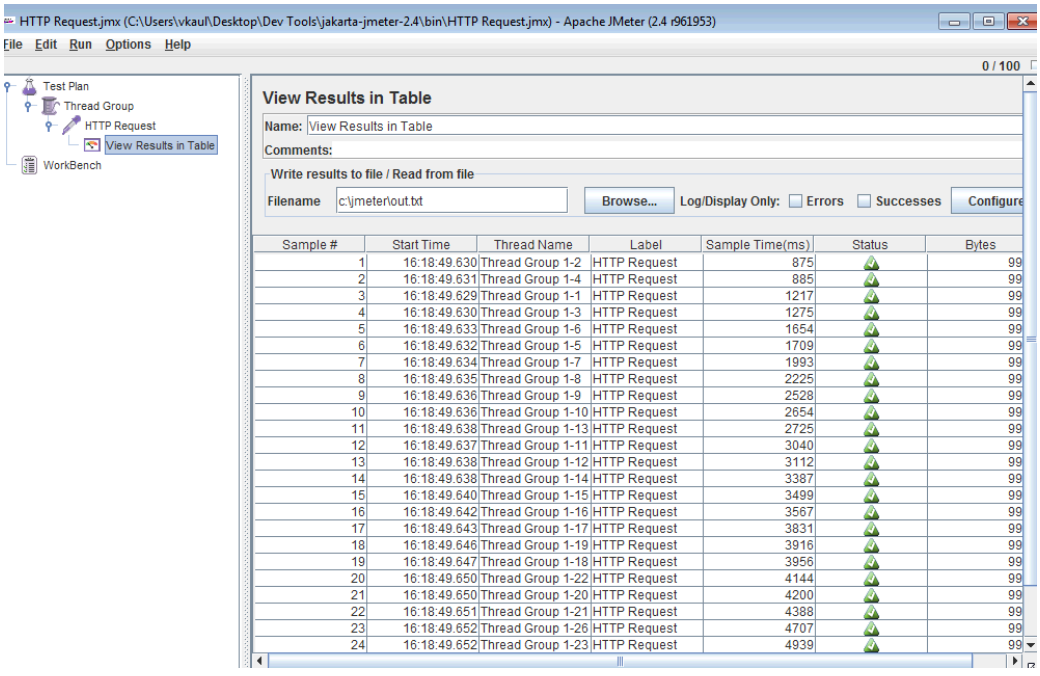


The final test plan will look something like this:



Step 5 : Running the test Case from GUI

We can run the test case by simply choosing “start” from the run menu or by pressing “CTRL+R”



Once this is done we can save the test as a jmx file and this can be invoked from the command line. Please note that the results along with being displayed on the screen will also be stored in an XML file, which the TestRig implementation will parse to see the performance.

A sample XML node for a single request:

<httpSample t="2719" lt="2719" ts="1294089192171" s="true" lb="HTTP Request" rc="200" rm="OK" tn="Thread Group 1-3" dt="text" by="9902"/>

Operation Contract Implementation

The operation contract for this will be very similar to the first example. This is because JMeter supports a rich command line interface. We will use all the methods in the same way but instead of using Java command line calls we will use:

jmeter -n -t mytest.jmx -l log.jtl

Once this is called we will simply use the **parseFile** to parse the given xml above and see if the performance is as expected. If so, we pass the test else we fail the test case.

It may be noted that performance tests are seldom passed or failed. Rather, they are analysed. In order to do this we would simply open the log.jtl file in Jmeter and see the results.

Outsafe – A Distributed database system in Amazon Cloud

Context

"Imagine an organization called Enterprise that wants to own and maintain a database that is robust to hardware and fail-stop software failures and that is distributed across many client sites. Enterprise would like to avoid ensuring (using in-house facilities) fault tolerance, durability, and distributed transactions because those features require a level of hardware infrastructure (dual backup sites, special disk configurations) and skill set (reliability experts) that Enterprise cannot afford. So, Enterprise

engages an outsourcing organization SupposedlyHonest (quite likely a Cloud owner) to support those features. The trouble is that Enterprise doesn't entirely trust SupposedlyHonest. Enterprise worries about inside hackers, outside hackers, and other customers of SupposedlyHonest.

OutSafe provides the following guarantees:

1. Enterprise clients can use simple single user database management systems locally. All operations on databases are possible including single record retrievals, complex queries, inserts, deletes, and updates. There is no requirement that the data accesses even be relational. Arbitrary transactions on data are allowed.

2. SupposedlyHonest will ensure ACID properties or will be shown to be cheating.

3. SupposedlyHonest will see only appends of encrypted data.

4. In access private mode, SupposedlyHonest will learn nothing about the access patterns of users. That is, if a user reads data item x, software at SupposedlyHonest will not know this. If user u1 accesses x and u2 accesses x, SupposedlyHonest won't know that they access the same data.

5. In private cloud mode, some client data will be access private and some will be in the supposedly private cloud. Transactions can mix the two without revealing accesses to the access private data. This mode is for databases requiring higher performance or having large data objects such as images.

The system performs roughly 8,000 TPC-C transactions per minute in access private mode, never aborts transactions, and is wait-free in access private mode and altruistically wait-free (as we will describe) in private cloud.

OutSafe also supports a ``Consultant-free" variant that requires no changes to application code to be made secure and distributed.

OutSafe is a deployable solution to the secure database-outsourcing problem.

In order to test the system, we need multiple systems. We need a system to act as a Conduit, another one as a driver and multiple clients. In order to do this we are using the Amazon Elastic Compute Cloud. It is a cost effective way of getting pre-configured machines in the cloud. These machines can be loaded with pre-configured images that contain all the data and settings required for the test case. EC2 also allows us to set up subnets within the machines that we have requested." [8]

Amazon EC2 exposes Web Service methods that can be used to control all this. What this gives us is an opportunity to automate the whole process. Let us look at the steps involved in running the Test Cases:

1. Request Amazon EC2 for N instances (N>=4) by specifying the AMI, instance type, request type and the spot price.

2. Wait for the instances to completely boot and then get each instance's external and internal IP.

3. Once the IP's have been collected, we need to write them to a file.

4. Then we need to make some changes to certain shell scripts. We need to insert the IP of the driver in the script.

5. Once all this is done, we transfer the IP file to the conduit and the scripts to their required location in the instances.

6. Once the scripts are in place, we simply have to start the server first and then call another Unix script that executed the DBT 2 scenarios.

7. Wait for the time entered by the user and then pull down the result files.

8. Parse files for results and make assertions.

Since this is a complex set of test cases we have designed a web application that controls all this. The web application has been designed in Java Struts. We will not go into details of the struts implementation. But we will see how all the steps have been implemented.

Operation Contract Implementation

In this section we will look at how the operation contract methods map to the steps listed above.

- The first method we look at is **initSytem.** This will take care of steps 1 and 2 listed above. This involves sending in a request for the instances and waiting for them to boot up to get their IP addresses. Let us look at the method that does this.

```
AmazonEC2 service = new AmazonEC2Client(accessKeyId, secretAccessKey);
List<String> secGroups=new ArrayList<String>();
secGroups.add("outsafe");
secGroups.add("default")'
RequestSpotInstancesRequest request = new RequestSpotInstancesRequest();
request.setSpotPrice(spotPrice);
request.setType(requestType);
request.setInstanceCount(Integer.parseInt(instanceCount));
request.setLaunchGroup(launchGroup);
LaunchSpecification spec=new LaunchSpecification();
spec.setImageId(ami);
spec.setInstanceType(instanceType);
spec.setSecurityGroup(secGroups);
request.setLaunchSpecification(spec);
RequestSpotInstancesResponse response = service.requestSpotInstances(request);
```

At this time we can parse the response and get the data we need. The complete code listing can be found in the TestRig project.

- The next method we use is **movefiles.** In this method we will make the appropriate changes and copy files over using the external IP addresses we got in the earlier steps. We implement that using SCP.

```
String command="scp -i /home/scripts/outsafe_key.pem "+source+" root@"+dns+":"+location;
Runtime runtime=Runtime.getRuntime();
                String output="";
                try {
                        Process p=runtime.exec(command);
                        String line="";
```

```
                BufferedReader outCommand = new BufferedReader(new
                                InputStreamReader(p.getInputStream()));
                while((line=outCommand.readLine())!=null)
                {
                        output+="<br>"+line;
                }

        } catch (IOException e) {
                // TODO Auto-generated catch block
                return e.toString();
        }
```

- Once the files have been moved over we need to run scripts. We do this in the **executeProcess** method.

```
        String executeCommand="ssh -i /home/scripts/outsafe_key.pem root@"+dns+" '"+scriptName+"'";

Runtime runtime=Runtime.getRuntime();
                String output="";
                try {
                        Process p=runtime.exec(command);
                        String line="";
                        BufferedReader outCommand = new BufferedReader(new
                                        InputStreamReader(p.getInputStream()));
                        while((line=outCommand.readLine())!=null)
                        {
                                output+="<br>"+line;
                        }

                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        return e.toString();
                }
```

- At this time all we need to do is use the **processWait, getResults** and **parseResults** to make sense of the results before we use **assertConclusion** to pass or fail the test.

The complete Struts application can be found in the TestRig project itself.

Appendix A - Interface

```
/**
 *
 */
package edu.nyu.testrig.contract;

/**
 * @author Vaibhav Kaul
 * vk565@nyu.edu
 *
 */
public interface TestRigOperationContract {

        /**
         * This method is used to initialize the system to be tested.
         * This could involve copying files, starting or stopping services etc.
         * The method returns a boolean to indicated if the initialization was successful or
not.
         * @param Array of Objects
         * @return boolean
         * @throws Exception
         */
        public boolean initSytem(Object[] o) throws Exception;

        /**
         * This function will allow users to move files. This could be
         * as trivial as copying files from a directory to another one
         * or maybe as complex as using FTP to put files on the cloud.
         * @param fileNames
         * @param source
         * @param destination
         * @return boolean
         * @throws Exception
         */
        public boolean movefiles(String[] fileNames, String[] source, String[] destination)
        throws Exception;

        /**
         * This function will allow a user to execute a process from within java.
         * This can be extremely useful for testing software that supports command line
interface.
         * @param executablePath
         * @param parameters
```

```
     * @return boolean
     * @throws Exception
     */
    public boolean executeProcess(String executablePath, String parameters) throws
Exception;

    /**
     * This function maybe useful in test cases with wait condition.
     * It could simply be used to cause the thread to sleep and give up
     * processor cycles for other processes or maybe in a load test cause
     * the thread to spin on a lock.
     * @param duration
     * @throws Exception
     */
    public void processWait(long duration) throws Exception;

    /**
     * This function is used to pull results of the test site.
     * @param o
     * @return
     * @throws Exception
     */
    public boolean getResults(Object[] o) throws Exception;

    /**
     * This function can be used to make sense of the results that were pulled down
     * @param o
     * @return
     * @throws Exception
     */
    public Object[] parseResults(Object[] o) throws Exception;

    /**
     * This should be used to assert the pass/fail condition
     * @param o
     * @throws Exception
     */
    public void assertConclusion(Object[] o) throws Exception;

    /**
     * Explicitly fails the test
     */
    public void failTest();
```

```
        /**
         * Explicitly passes the test
         */
        public void passTest();

        /**
         * To run the clean up script.
         * @param o
         * @return
         */
        public Object[] cleanup(Object[] o);
}
```

References

[1] Software Development Process - http://en.wikipedia.org/wiki/Software_development_process

[2] Automation Testing Tools - http://en.wikipedia.org/wiki/Test_automation

[3] ATF – Agile Testing Framework, Developed by the Author while working at Sapient Corporation.

[4] How Selenium Works Diagram - http://seleniumhq.org/about/how.html

[5] Selenium ID methods : http://release.seleniumhq.org/selenium-remote-control/0.9.0/doc/java/

[6] WCF - http://en.wikipedia.org/wiki/Windows_Communication_Foundation

[7] JMeter - http://jakarta.apache.org/jmeter/

[8] Outsafe - The text is a part of the abstract of an unpublished manuscript by Meacham and Shasha.