Int. J. Advanced Networking and Applications
Volume: 03, Issue: 04, Pages:1261-1269 (2012)

Validation of Internet Application: Study, Analysis and Evaluation

Dinesh Kumar
Asst. Professor,Shri Siddhi Vinayak Institute of Technology, Bareilly,
Email: mail2dinesh.gm@gmail.com

----------ABSTRACT----------

Today, testing applications for Internet (web sites and other applications) is being verified using proprietary test solutions. The Internet Security became a very important and complex field of researches in our present time, especially if we apply this to the discussion of Internet protocols as basic interfaces for exchanging sensitive data over the Internet and finding appropriate and trustworthy algorithms for their validation. Test Competence Centre at Ericsson AB has expertise on testing telecom applications using TTCN-2 and TTCN-3 notations. These notations have lot of potential and are being used for testing in various areas. So far, not much work has been done on using TTCN notations for testing Internet application. This thesis was a step through which the capabilities/possibilities of the TTCN notation (in Web testing) could be determined. This paper presents investigation results of the 3 different test Technologies/tools (TTCN-2, TTCN-3 and a proprietary free software, PureTest) to see which one is the best for testing Internet Applications and what are the drawbacks/benefits each technology has.

Keywords - **validation, web testing, tools**

Date of Submission: August 26, 2011 Revised: October 12, 2011 Date of Acceptance: October 28, 2011

I. INTRODUCTION

An ever increasing number of users are becoming dependent on Internet services, such as search engines, e-mail, and music jukeboxes for their work and leisure. These services typically comprise complex conglomerates of distributed hardware, software, and databases. Thus, ensuring high service availability is challenging ([1], [2]). Our work seeks to alleviate one important source of service failures: *operator mistakes*. Several studies have shown that mistakes are a significant source of unavailability [1]. [5]. For instance, Oppenheimer et al. [1] show that mistakes were responsible for 19-36% of failures, and, for 2 out of 3 services, were the dominant source of failures and the largest contributor to time to repair. Similarly, Oliveira et al. [5] report that operator mistakes are responsible for a large fraction of the problems in database administration. Both corroborate an older study of Tandem systems where mistakes were a dominant reason for outages [3]. In our previous work, we proposed operator action *validation* as an approach for detecting mistakes while hiding them from the service and its users ([5], [6]). In this approach, a validation framework creates an isolated extension of the online service in which operator actions are performed and later validated. Before the operator acts on a service component, the component is moved to this extension. After the operator .If validation succeeds, the system moves the component back online; otherwise, it alerts the operator. While this validation strategy can detect and hide a large class of mistakes, it has three important limitations: (1) it requires known instances of correct behavior for comparison; (2) it provides no guidance in pinpointing mistakes; and (3) it fails to detect latent mistakes. In this paper, we propose a novel validation strategy, called *model-based validation* that addresses these limitations. Model-based validation calls for service engineers1 to choose abstract models to describe the systems and identify incorrect configurations and behaviors. These models are then used to guide the specification of assertions to check the correctness of operator actions without requiring instances of correct behaviors for comparison. The purpose of the models is to ensure a systematic and proactive approach to generating assertions, rather than an ad-hoc/reactive approach that may leave many mistakes undetected. The use of Internet has grown over the years. Communication and commerce through Internet has become a central focus for businesses, consumers, government and the media. In this environment it is a must that the web site/ application performs the way it is supposed to. Therefore thorough testing is needed before releasing the application/site on the web. Test Competence Centre at Ericsson AB has, for many years, testing telecom applications (using TTCN-2 and TTCN-3) as a key area of expertise. So far not much work has been done on testing Internet applications using TTCN-2 or TTCN-3 languages. The department wishes to broaden its knowledge in this area and the Master Thesis is a step taken in that direction. Testing Internet Applications is a very complex task and not everything can be tested in a short duration of time. Therefore, the supervisor at Ericsson AB identified certain criteria on

Volume: 03, Issue: 04, Pages:1261-1269 (2012)

which the thesis work would concentrate. Some of the identified areas where:

- Testing the websites for broken links
- Identifying all the resources in the web site
- Calculation of Server response time
- Automated website testing

Based on these criteria, search for a third technology (apart from TTCN-2 & TTCN-3), a proprietary tool, was conducted during the thesis work and it was found that Pure Test from Minq Software AB [9] fits the above criteria more precisely than other tools.

The three technologies are:

TTCN2: This is the current version for TTCN and has been in use for a long time. This is a stable technology and aimed for verification of protocol conformance.

TTCN3: The coming technology is TTCN-3 that is intended to be more user friendly and aimed for a wider test audience.

The last technology is to use proprietary free software for verification. This is not a standard technology but may fit the needs well and is currently the normal way to verify Internet applications.

The research work was a way to gain/improve knowledge in areas like:

- Increase Knowledge of TTCN-2 notation
- Usage of TTCN-2 tool for testing
- Developing an adaptation for HTTP testing with TTCN-3 (C++ design)
- Increase Knowledge of TTCN-3 notation
- Usage of TTCN-3 tool for testing
- Usage of a tool for Internet application for testing
- Selected Internet Application and related protocols.

The Internet Security became a very important and complex field of researches in our present time, especially if we apply this to the discussion of Internet protocols as basic interfaces for exchanging sensitive data over the Internet and finding appropriate and trustworthy algorithms for their validation. The core idea of validation is to verify operator actions under realistic workloads in a realistic but isolated validation environment [6]. Mistakes can then be caught before becoming visible to the users. To achieve realism, the validation environment is hosted by the online system itself particular, a service with validation is divided into two slices, an online slice that hosts online components and a validation slice where components can be operated on and validated before being re-integrated into the online slice. The validation slice contains a testing harness that can be used to load the components. Validation proceeds as follows.: Suppose an operator needs to operate on a service component (e.g., to upgrade its software). Before starting, the operator uses a script to move the server hosting the component from the online slice to the validation slice. The operator can now work on the component without affecting the online system. After completing her task, the operator surrounds the masked component with proxies that give the illusion that the masked component is in a complete system. She then places a validation workload on the masked component. Validation compares the replies of the masked component with those in the trace or those of the online component. If the replies match (according to content-similarity and performance criteria), the framework considers the operator actions to be validated and moves the hosting server node back online. If validation fails, the system alerts the operator.

Validation is designed to address a serious issue in traditional testing (which we call *offline testing*). Thus, even with careful testing, operators can make mistakes when changing or deploying their changes to the online system. Validation closes this gap between offline testing and the online system, although the two approaches can be complementary: validation could be applied as the last step in a testing/validation process before exposing an operator action to the online system. We also proposed a primitive version of model-based validation in [5]. The expected schema then represents the model against which the actions are validated. Here, we extend our original proposal significantly by applying model-based validation to entire Internet services. The rest of this paper is organized as follows. In the first part of this paper, we introduce the concept about validation of internet applications . The next section includes testing and its types it also describes classification of web testing and next section give elaborative description of TTCN. Remainder of the paper contains evaluation and result analysis leading to final conclusion

2. TESTING

Software Testing is the process of executing a program or system with the intent of finding errors. Or, it involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. Software is not unlike other physical processes where inputs are received and outputs are produced. Where software differs is in the manner in which it fails. Most physical systems fail in a fixed (and reasonably small) set of ways. By contrast, software can fail in many bizarre ways. Detecting all of the different failure modes for software is generally infeasible. Unlike most physical systems, most of the defects in software are design errors, not manufacturing defects. Software does not suffer from corrosion, wear-and-tear -- generally it will not change until upgrades, or until obsolescence. So once the software is shipped, the design defects -- or bugs -- will be buried in and remain latent until activation. Software testing has three main purposes: verification, validation, and defect finding.

♦ the *verification* process confirms that the software meets its technical specifications. A “specification” is a description of a function in terms of a measurable output value given a specific input value under specific preconditions. A simple specification may be along the line of “a SQL query retrieving data for a single account against the multi-month account-summary table must return these eight fields <list> ordered by month within 3 seconds of submission.”

♦ the *validation* process confirms that the software meets the business requirements. A simple example of a business

Volume: 03, Issue: 04, Pages:1261-1269 (2012)

requirement is "After choosing a branch office name, information about the branch's customer account managers will appear in a new window. The window will present manager identification and summary information about each manager's customer base: <list of data elements>." Other requirements provide details on how the data will be summarized, formatted and displayed.

♦ a *defect* is a variance between the expected and actual result. The defect's ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

2.1 Testing Process

Many people believe that testing is only what happens after code or other parts of a system are ready to run. They assume that testing is only test execution. Thus, they don't think about testing until they're ready to start executing tests. Testing is more than tests. The testing process also involves identifying what to test (test conditions) and how they'll be tested (designing test cases), building the tests, executing them and finally, evaluating the results, checking completion criteria and reporting progress.

First, test what's important. Focus on the core functionality—the parts that are critical or popular—before looking at the 'nice to have' features. Concentrate on the application's capabilities in common usage situations before going on to unlikely situations. It's worth saying again: focus on what's important.

The value of software testing is that it goes far beyond testing the underlying code. It also examines the functional behavior of the application. It's entirely possible that the code is solid but the requirements were inaccurately or incompletely collected and communicated. It's entirely possible that the application can be doing exactly what we're telling it to do but we're not telling it to do the right thing. A comprehensive testing regime examines all components associated with the application. Even more, testing provides an opportunity to validate and verify things like the assumptions that went into the requirements, the appropriateness of the systems that the application is to run on, and the manuals and documentation that accompany the application. If we leave test design until the last moment, we won't find the serious errors in architectural and business logic until the very end. By that time, it becomes tricky and expensive to track and fix these faults from the whole system.

2.2 Test Phases

The figure below shows the relationship between different phases of software developed and testing. The relationships between the phases are based on the V-model, as presented by [1].

2.2.1 The V-MODEL of software testing Software testing is too important to leave to the end of the project, and the V-Model (fig.1)of testing incorporates testing into the entire software development life cycle. In a diagram of the V-Model, the V proceeds down and then up, from left to right depicting the basic sequence of development and testing activities. The model highlights the existence of different levels of testing and depicts the way each relates to a different development phase. Like any model, the V-Model has detractors and arguably has deficiencies and alternatives but it clearly illustrates that testing can and should start at the very beginning of the project. In the requirements gathering stage the business requirements can verify and validate the business case used to justify the project. The business requirements are also used to guide the user acceptance testing. The model illustrates how each subsequent phase should verify and validate work done in the previous phase, and how work done during development is used to guide the individual testing phases. This interconnectedness lets us identify important errors, omissions, and other problems before they can do serious harm. On the development side, development cycle is started by defining business requirements. These requirements are then translated into high- and low-level designs, and finally implemented in program code (a unit). On the test execution side, unit tests are executed first, followed by integration, system and acceptance tests. Below is the brief description of the different test phases.

2.2.2 Unit Test

Starting from the bottom the first test level is 'Unit Testing'. Unit tests focus on the types of faults that occur when writing code, such as boundary value errors in validating user input. A series of stand-alone tests are conducted during Unit Testing. Each test examines an individual component that is new or has been modified. A unit test is also called a module test because it tests the individual units of code that comprise the application. Each test validates a single module that, based on the technical design documents, was built to perform a certain task with the expectation that it will behave in a specific way or produce specific results. Unit tests focus on functionality and reliability, and the entry and exit criteria can be the same for each module or specific to a particular module. Unit testing is done in a test environment prior to system integration. If a defect is discovered during a unit test, the severity of the defect will dictate whether or not it will be fixed before the module is approved. The problem with a unit is that it performs only a small part of the functionality of a system, and it relies on co-operating with other parts of the system, which may not have been built yet.

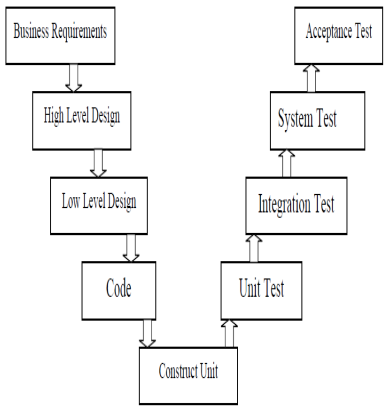

Fig. 1: Test phases in relation with development phases V-Model

Volume: 03, Issue: 04, Pages:1261-1269 (2012)

2.2.3Integration Testing

Integration testing examines all the components and modules that are new, changed, affected by a change, or needed to form a complete system. Where system testing tries to minimize outside factors, integration testing requires involvement of other systems and interfaces with other applications, including those owned by an outside vendor, external partners, or the customer. Integration tests focus on low-level design. They check for errors in interfaces between units and other integrations. As the components are constructed and tested they are then linked together to check if they work with each other. It is a quite possible that the two components that have passed all their tests, when connect to each other, produce a new component full of faults. These tests can be done by specialists, or by the developers.

2.2.4 System Testing

System tests check whether the system as a whole implements effectively the high-level design. Once the entire system has been built, it has to be tested against the "System Specification" to check if it delivers the features required. System Testing tests all components and modules that are new, changed, affected by a change, or needed to form the complete application. The system test may require involvement of other systems but this should be minimized as much as possible to reduce the risk of externally-induced problems. Testing the interaction with other parts of the complete system comes in Integration Testing. The emphasis in system testing is validating and verifying the functional design specification and seeing how all the modules work together. For example, the system test for a new web interface that collects user input for addition to a database doesn't need to include the database's ETL application—processing can stop when the data is moved to the data staging area if there is one. System Testing is not about checking the individual parts of the design, but about checking the system as a whole.

2.2.5 Acceptance Testing

Acceptance tests are ordinarily performed by the business/users to confirm that the product meets the business requirements. Acceptance Testing checks the system against the "Requirements". User Acceptance Testing is also called Beta testing, application testing, and end-user testing. Whatever you choose to call it, it's where testing moves from the hands of the IT department into those of the business users. Software vendors often make extensive use of Beta testing, some more formally than others, because they can get users to do it for free. The customer should always do acceptance testing. The developers should not do this testing. The customer knows what is required from the system and is the only person qualified to make that judgment.

2.3 Web Testing

Web applications are based on client-server, request-response mechanisms. At a high level, Web applications are usually divided into four basic layers. Layers 3 and 4 are optional and are chosen

Based on product requirements:

1. Presentation layer (client side/user interface)
2. Distribution layer (server side)
3. Business logic layer
4. Back end (database/external dependency)

The general flow of this architecture is as follows: The client (presentation layer) will request a URL. A Web server (distribution layer) will receive the request and carry the preliminary processing. Based on processing, the Web server will call the business logic layer. The business logic layer carries out further processing based on encapsulated business rules. The business logic will also interact with back-end database applications (persistence layer) as well as any external applications. The business logic will return control to the Web server when processing completes. The Web server will send a response to the client.

2.3.1 Internet applications

The title of this paper, Validation of Internet Applications, creates a need to define what we mean with web applications in this thesis work. The two author's classifications of web sites were presented in the above section. Powell [2], in his classification does not use the word 'application' until a higher degree of interactivity is offered. Instead he uses the word 'Site' for the first, simpler, categories. For this thesis work, we have considered web application as any web based site or application available on Internet or on an Intranet, whether it is a static promotion site or a highly interactive site for banking services. However, due to time constraints the testing is done only on the static website. In this thesis work, web site and web application have the same meaning.

2.5 Test types

The previous text describes the general guidelines for testing, whether it is software applications or web applications. But the scope of this thesis is testing web applications. There are different types of tests that are performed within the different stages throughout the web testing process. Below text describes briefly the most common web site/application test types used, with aim on the medium of the web.

Functionality testing

Functionality testing is one of the most important areas of testing. It should never be missed. Functionality testing involves an assessment of every aspect of the site where scripting or code is involved, from searching for dead links, to testing forms and scripts. The purpose of this type of test is to ensure that every function is working according to the specifications. Functions apply to a complete system as well as a separated unit.

Browser Compatibility

There are a number of different browsers and browser options. A website has to be designed to be compatible for a majority of the browsers. This still leaves room for creativity. Even with Microsoft's Internet Explorer and Netscape's Navigator this is an issue because of the different versions people are or still are using

Performance testing

Volume: 03, Issue: 04, Pages:1261-1269 (2012)

Performance testing generally describes the processes of making the web site/application and its server as efficient as possible, in terms of download speed, machine resource usage, and server request handling. In order to identify bottlenecks, the system or application has to be tested under various conditions. Varying the number of users and what the users are doing helps identify weak areas that are not shown during normal use.

Transaction Testing

This is very critical in an e-business application. The software a website is utilizing has to be forced to invoke its various components and whether the direct and indirect interfaces work correctly. The information entered by the user should make it to the database in the proper ways. When the user calls for information contained in the database, the proper data must be returned

Usability

Usability testing is the process by which the human-computer interaction characteristics of a system are measured. The measurement shows the weaknesses, which leads to correction. To ensure that the product will be accepted on the market it has to appeal to users. There are several ways to measure usability and user response.

Compatibility testing

Compatibility testing measures how well pages display on different clients. For example: browsers, different browser version, different operating systems, and different machines. This testing is sometimes also referred as browser compatibility testing or cross-browser testing.

Security testing

Security testing refers to the testing of the site and web server configuration with an eye towards eliminating any security or access loopholes. In order to persuade customers to use Internet banking services or shop over the web, security must be high enough. One must feel safe when posting personal information on a site in order to use it. Typical areas to test are directory setup, SSL, logins, firewalls and log files.

3. TTCN

TTCN (Tree and Tabular Combined Notation for TTCN-2, or Testing and Test Control Notation for TTCN-3) is a globally adopted standard test notation for the specification of test cases. A TTCN specified test suite is a collection of test cases together with all the declarations and components needed for the test. Its use has grown considerably since its first launch and it is used in many fields such as:

• **Telecommunication networks**

GSM, ISDN, 3GPP/UMTS, TETRA etc.

• **Telecommunications systems**

Public exchanges, private branch exchanges, terminal equipment like (mobile) handsets,

Fax machines, PC communications cards

• **Telecommunications interfaces/protocols**

INAP, ISUP, SS7, ATM, Voice over IP, Wireless LANs (Hiperlan/2), UMTS/3G etc.)

TTCN was an initiative of ETSI, the European Telecommunications Standards Institute. It is Internationally standardized by International Organization for Standardization (ISO). The first published standard of TTCN was released in 1992. The language is now supported by a large variety of sophisticated tools such as test systems, editors, compilers, syntax checkers and simulators. TTCN is an abstract language; abstract in the sense that it is test system independent. This means that a test suite in TTCN for one application can be used in any test environment for that application.

3.1 TTCN-2

TTCN-2 is used worldwide to define standards. It is, for example, often used by ETSI for the definition of conformance test suites for telecom standards, e.g. GSM, DECT, ISDN, and TETRA. Most recently, it has been the language of choice for testing of Bluetooth and UMTS. Telecom companies developing products use TTCN to test whether their product will function according to the standard. TTCN is not only used in standardization work. The language is very suitable for conformance testing of real-time and communicating systems. This has led to a wide usage throughout the telecommunications industry. TTCN can also be used outside the telecommunications field, for conformance testing of communicating systems or protocols. These test suites describe black box tests for reactive communication protocols and services. It is a standardized notation which supports the specification of abstract test suites for protocol conformance testing. An abstract test suite is a collection of abstract test cases which contains all the information that is necessary to specify a test purpose.

3.1.1 Basic Notation

A TTCN test suite consists of following parts:

• *Overview Part:*

The overview part of a TTCN test suite is like a table of contents. It provides all information needed for the general presentation and understanding of the test suite. It states the test suite name and test architecture, describes the test suite structure, if any additional documents related to the test procedure is available, it provides references to them and includes indexes for the test cases, test steps and default behavior descriptions.

• *Declarations Part:*

The declarations part provides definitions and declarations used in the subsequent parts of the test suite. The declarations part declares types, test suite operations, selection expressions, test components, PCOs, ASPs, PDUs, timers and variables. The constraints part of a TTCN test suite provides the values of the PDUs and ASPs to be

• *The Dynamic Part*

The dynamic part describes the dynamic behavior of the test processes by test cases, test steps and default behavior descriptions. A test case is like a complete program, which has to be executed in order to judge whether a test purpose is fulfilled or not. Test cases can be structured/divided into test steps and default behavior descriptions. A test step can be seen as a procedure definition, which can be called in test cases.

3.2 TTCN-3

TTCN-3 is the current version of TTCN and has recently been standardized by ETSI (European Telecommunication Standards Institute) and by ITU (International Telecommunication Union). It is a complete redesign and widens the TTCN application area to different kinds of testing applied to different technologies in the telecommunication and IT domain in general. The syntax of the textual TTCN-3 core notation is like a programming languages and similar to C++ or Java. No graphical editors are required for the core notation, but could be used if the graphical format of TTCN-3 is applied instead. Because TTCN-3 is a totally new technique, it requires new tool support. First parsers, compilers, run-time environments and editors are available [13] and have been used for this work. With TTCN-3 the existing concepts for test specifications have been consolidated. Besides consolidation, TTCN-3 defines new concepts to widen the scope of applicability of it. TTCN offers the possibility to produce test cases for telecommunications networks, systems and interfaces independently of the underlying test system hardware and software**.**

3.2.1 Basic Notation

This section presents a brief introduction of TTCN-3 language. Detailed description of the notation can be found in TTCN-3 standard [4]. The top-level unit of a TTCN-3 test suite is a module, which can import definitions from other
Modules. A module consists of a declarations part and a control part.

```
module My Module {
import from MyDeclarations;
control {
    execute(MyTestCase());
} // end control
} // end module MyModule
```

The declarations part of a module contains definitions, e.g., for test components, their communication interfaces (so called ports), type definitions, test data templates, functions, and test cases. The control part of a module calls the test cases and describes the test campaign. The imported module contains the definition for the test case MyTestCase, which is performed in the control part with the execute statement.

4. TOOL DESCRIPTION

This chapter presents the tools (TTCN-2, TTCN-3 & Pure Test) that were used for this thesis work. Firstly, the TTCN-2 tool is presented along with brief introduction of its sub-tools, followed by description of the TTCN-3 tool set and Pure Test. Finally, a description of other tools is given.

4.1 TTCN-2 Tools

TTCN Basic is a package that consists of the Ericsson made SCS (System Certification System)
and the ITEX editor from Telelogic [6]. Since SCS could be used with any other editor, and could
be bought separately from the ITEX editor, I have chosen to just describe SCS here.

Here is the basic introduction of different parts of TTCN test system:

TTCN Editor

TTCN test suites can be written using graphical editors. There are various editors available in the market but Ericsson uses ITEX which is most commonly used.

TTCN Launcher

A GUI for launching different tools and test configuration. It maintains different projects and tools configuration.

TTCN Manager

It is the main tool for test suite execution. It has a graphical user interface from which all functions are accessed. It provides easy access to others tools like TTCN Editor, TTCN Translator, TTCN Executor and Log Monitor.

TTCN Translator

It is a compiler which converts the TTCN machine processable format (.mp) into an internal format Executable Test Language(ExTeL). Translator can take multiple TTCN test suites and generates ExTeL files for each one of them.

TTCN Executor

It executes the ExTeL files produced by translator and produces log files. During execution, the executor communicates to Implementation Under Test via the test port.

Test Port

The responsibility of the test port is to take care of the communication between the TTCN executor and the interface towards the Implementation under test.

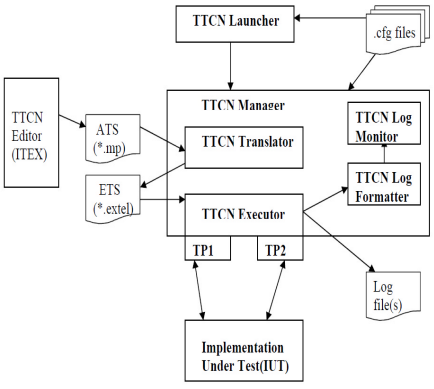

Fig. 2:TTCN-2 test system

4.2 TTCN-3 Tools

Titan, an internally developed tool by Ericsson, is a Ericsson wide official tool set for TTCN-3 There are various other TTCN-3 tool set vendors such as

Here is the basic introduction of different parts of TTCN-3 test system :

TTCN-3/ASN.1 Compiler

Translates TTCN-3 and ASN.1 modules into C++ programs. Each module is converted into one C++ header and source file.

Base Library

This library is written in C++ and provides important supplementary functions for the generated code.

Test Port

A test port skeleton is generated by the TTCN-3/ASN.1 compiler. The user can then implement the functionality required to communicate with the system under test. The responsibility of the test port is to take care of the communication between the TTCN-3 Test system and the system under test. Please see section for more detailed description on test ports.

5. TEST PORT

As one of the task in this thesis work was to implement a test port for HTTP protocol for TTCN-3 execution system, let us look at the concept of test port in detail. The concept was introduced in the SCS tool and is also adopted by the TTCN-3 executor(fig.3).

5.1 Concept

The goal of the TTCN test system is to make it possible to execute TTCN Test Suites towards any

interface (internal or external) in any system

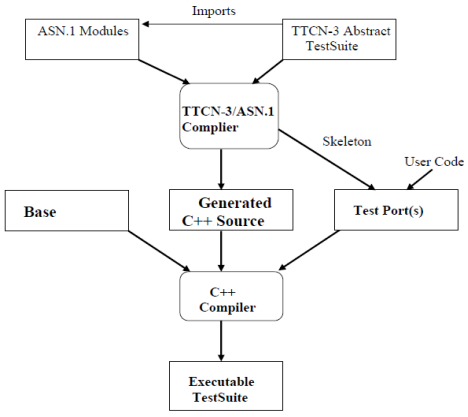

Fig. 3:Structure of TTCN-3 EXECUTER

Both TTCN-2 and TTCN-3 achieve this goal by including a possibility of adding new interface adapters to the system without having any need to change the core executor. A test port consists of two parts(fig. 4). The first part is an adaptation to the Tools(SCS Tools/Titan Tools) which is used when it's linked into the Tools.

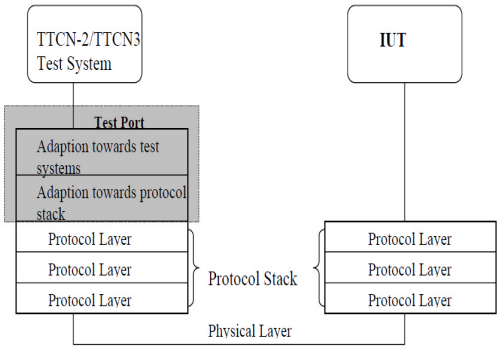

Fig. 4: Conceptual view of test port

1. EVALUATION

Based on the criteria described in Chapter 1.1, Test Cases were written using TTCN-2 and TTCN-3. In the case of Pure Test, it was just to run the tool against the site under test. It was seen during the work that any kind of information/results that Pure Test presented, it was possible to do the same using TTCN. During the work, the main concentration was on TTCN-3 test case design as it was seen that it was possible to design same test cases in TTCN-2 as in TTCN-3. So in some cases, comparison is done only between TTCN-3 and Pure Test.

6.1 Accuracy

Tests were run on an Ericsson internal website

a. http://esekant027.epk.ericsson.se/042/

On manual count, the total number of resources in the above website was 67, out of which 3 were erroneous.

From the Table 1, it can be seen that TTCN-3 tests were better than Pure Test when compared to the actual manual count(67 resources with 3 errors). The main reason for it is the performance of Html Parser. TTCN-3 Html Parser is more accurate in finding the resources because during the testing,

some errors were detected and fixed, making it a better parser as time progressed. One such HTML convention, which Html Parser in both TTCN and Pure Test was unable to handle, was a url name containing '&' This stands for 'ampersand' in HTML. While fetching the urls containing this, the application have to change it to '&'.

Given below are the results of the tests performed on several other internal websites of Ericsson.

	TTCN-3 Result	PureTest results
Total Pages	62	50
Total size	3625 kB	3396 kB
Cacheble Pages	62	46
Private Pages	0	0
Non-Cacheble Pages	0	0
Content Type		
Text/html	40	30
Images	13	7
Pdf		0
Multimedia		0
Others	9	9
Error	3	4
Total Time	3.2 s	5.0 s

Table 1: Result Comparison of TTCN-3 and Pure Test

b. http://esekant027.epk.ericsson.se/019/(table2)

	TTCN-3 Result	PureTest results
Total Pages	172	172
Total size	2386 kB	3538 kB
Cacheble Pages	172	172
Private Pages	0	0
Non-Cacheble Pages	0	0
Content Type		
Text/html	13	15
Images	159	156
Pdf	-	1
Multimedia	0	0
Others	0	0
Error	3	2
Total Time	207.0 s	6.0 s

Table 2: Result Comparison of TTCN-3 and Pure Test

Volume: 03, Issue: 04, Pages:1261-1269 (2012)

7. OPTIMIZATION

TTCN-3 HtmlParser is not very optimized as not much attention was given to it on this aspect. The main goal was to create a simple parser that could parse html body that gives back a set of resources in it. Due to time constraint, not much thought was put on optimizing it. Pure Test scores on this aspect as it is a proper product which has evolved during time and thus it is greatly optimized and robust when compared with the TTCN-3 HtmlParser. A page of size 50 kB parses in TTCN-3 as quickly as it does in Pure Test. A test was performed on a web page of size 330 kB. Pure Test took 3-4 seconds to parse the contents and get the resources

- *Speed*

If we see the execution time for tested websites (with smaller html pages), on an average TTCN-3 xecution takes less time to traverse through the whole website when compared to Pure Test. But
when testing a website containing big html pages, TTCN-3 test execution goes much slower than Pure Test due to the factor presented in Optimization section. Pure Test scores heavily over
TTCN-3 in this aspect.

- *User Interface*

Pure Test is a well-developed product. It has a very easy to use graphical user interface which makes it very easy to configure and operate. TTCN-3 on other hand is like a programming language in which test cases can be written using editors. The result of TTCN-3 test cases when compiled in the Titan tool is an executable program which runs from a command line with no graphical user interface. The results are presented in a log file, which has a textual format. Although the results can be exported to an excel sheet by writing some external functions using C++, it is not an easy task for a person with no programming background. Pure Test provides the results in graphs and structures, which are easy to understand and maintain.

- *Cost*

License fees for TTCN-2 and TTCN-3 tool sets are free of cost within Ericsson but the other tool vendors around the world charge hefty amount of money for annual licenses. Another cost related to these technologies is that if the project members do not have the knowledge of TTCN, it costs the company money and time to bring the workers to appropriate competence level. These are huge drawbacks when compared to Pure Test (or for that matter many proprietary software's). Pure Test is free software and being a simple application with a user-friendly interface, it is easy to configure and run thus saving the huge lead-time for the testing work.

7.1 Comparing TTCN-3 to TTCN-2

Having developed a test suite for the GIOP protocol in TTCN-3, TTCN-3 met the expectation of being directly and easily applicable for testing of message based systems. In particular, TTCN-3 showed several distinct advantages over TTCN-2. In another study, TTCN-3 has also been shown to be effective for the testing of operation-based interfaces, components and applications [9].
The advantages of using TTCN-3 for defining a GIOP test suite have been firstly in the easy use of the textual core notation of TTCN-3. For editing the abstract test suite, the GNU Emacs with a special TTCN-3 mode [18] has been used. That helped to develop the ATS in an integrated manner with traversing, syntax and semantics checking, and code generation capabilities efficiently. In addition, the test specification can be developed and is readable in
every text editor without having the need for cost-intensive development environments. The syntax of TTCN-3 showed to be suitable to specify GIOP messages relatively fast and well readable. In the ongoing work, tests for the client side of ORBs will be developed. With the module and import concepts of TTCN-3, significant parts of the existing test suite can be reused to specify and implement the additional tests. Even, the two test suites can be combined into one so that the test suite user has to perform only one set of test cases. Technically however, two different types of main test components will be used in that integrated test suite to reflect the different configurations of client and server side tests. An integrated test suite will be more practical and comfortable. By means of two external functions to start the CORBA client and to stop it after test execution, which are used in the control part, the tests can control that CORBA client automatically. Beyond the concrete technical advantages of TTCN-3 for the GIOP tests, there are further ones. In practice TTCN-3, requires less code than TTCN-2 to express the same test behavior.

8. CONCLUSION

With the experience gained in the three technologies during this thesis work, it can be said that the best technology for testing Internet Application is Pure Test. It has several advantages over the other two technologies. Some of the advantages are cost benefits, graphical user interface, fast parsing and robustness scores over the other two technologies. Of the other two technologies, TTCN-3 is certainly better than TTCN-2 as it has features like C++, Perl integration, regular expressions engine, closeness to programming languages making it easier to write complex test cases etc. This technology is still evolving and tool vendors are, and will be adding add-on features helping the technology to grow more. TTCN-2 is an industry proven technology for conformance, function testing etc. but it can be said that it is not as flexible as TTCN-3 for Internet Application testing. However, it was wonderful to see that TTCN-2 language and the tool set had ways (although not the very best or optimized) through which it was possible to make test scenarios similar to TTCN-3.

REFERNCES

[1]. John Clark and Jeremy Jacob. A survey of authentication protocol literature : Version 1.0., November 1997http://www-users.cs.york.ac.uk/jac/papers/drareview.ps.gz

[2]. Powell, Thomas A., Web Site Development, Beyond Web Page Design, 1998; Prentice Halls; ISBN 0-136-50920-7

[3]. Catherine Meadows: Formal Verification of Cryptographic Protocols: A Survey. ASIACRYPT 1994 Survey in Formal Analysis of Security Properties of Cryptographic Protocols, Tarigan 2002

[4]. D. Dolev, A. Yao, *On the Security of Public Key Protocols*, IEEE Trans. on Information Theory, 1983

[5]. Michael Burrows, Martin Abadi, and Roger Needham. Logic of authentication. Technical Report 39, Digital Systems Research Centre, February 1989 Protocol Verification by the Inductive Method,

[6]. Abadi/Gordon, 1998 [AR00] Reconciling two Views of cryptography (The Computational Soundness of Formal Encryption), Abadi/Rogaway, 2000 [AD94] A Theory of Timed Automata,

[7]. Alur/Dill, Theoretical Computer Science, 1994 [AJ04] Three Tools for Model-Checking Security Protocols,

[8]. Luca Vigano', Electronic Notes in Theoretical Computer Science, 2006 [AVISPA]

[9]. D. Oppenheimer *et al.*, .Why do Internet Services Fail, and What Can Be Done About It. in *USITS*, 2003.

[10]. D. A. Patterson *et al.*, .ROC: Motivation, Definition, Techniques, and Case Studies, UC, Berkeley, Tech. Rep. UCB//CSD-02-1175, Mar. 2002.

[11]. J. Gray, .Why does Computers Stop and What Can Be Done About It? in *SRDS*, 1986.

[12]. B. Murphy and B. Levidow, .Windows 2000 Dependability, Microsoft Research, Tech. Rep. MSR-TR-2000-56, June 2000.

[13]. F. Oliveira *et al.*, understanding and Validating Database System Administration, in *USENIX*, 2006.

[14]. K. Nagaraja *et al.*, understanding and Dealing with Operator Mistakes in Internet Services, in *OSDI*, 2004.[7] Rice University, .DynaServer Project,. http://www.cs.rice.edu/CS/Systems/DynaServer, 2003.

[15]. P. Anderson *et al.*, .Smart Frog Meets LCFG: Autonomous Reconfiguration with Central Policy Control, in *LISA*, 2003.

[16]. Y.-Y. Su *et al.*, .Auto Bash: Improving Configuration Management with Operating System Causality Analysis, in *SOSP*, 2007.

[17]. W. Zheng *et al.*, .Automatic Configuration of Internet Services, In *EuroSys*, 2007.

[18]. A. Whitaker *et al.*, .Configuration Debugging as Search: Finding the Needle in the Haystack, in *OSDI*, 2004.

[19]. M. K. Aguilera *et al.*, .Performance Debugging for Distributed Systems of Black Boxes, in *SOSP*, 2003.

[20]. P. Barham *et al.*, .Magpie: Real-Time Modeling and Performance-Aware Systems, in *HotOS IX*, 2003.