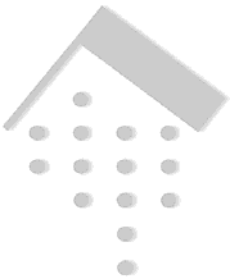


Fachhochschule Wiesbaden
University of Applied Sciences

Karlstad University
Sweden

Department of Computer Science

Nisar Lalani

Validation of Internet Applications

Master's Thesis
2005

This report is submitted in partial fulfillment of the requirements for the Master's Degree in Computer Science. All material in this report which is not my work has been identified and no material is included for which a degree has been previously conferred.

_____

Nisar Lalani

Approved : 2005-12-30

_____

Examiner : B.Dreher
Fachhochschule Wiesbaden - University of Applied Sciences

_____

Examiner : D.F.Ross
Karlstad University

Nisar Lalani

Validation of Internet Applications

Master's Thesis
2005

Acknowledgments

This thesis was written at Ericsson AB in Karlskrona, Sweden.

I would like to thank Prof. Dr. Björn Dreher for undertaking the supervision of this thesis and his support during the work.

I am thankful to my department manager Tor Nilsson for giving me the chance to write this thesis. My industrial supervisor, Tomas Nilsson deserves special thanks for his valuable proposals, ideas and encouraging attitude during the writing of the thesis.

Finally I would like to thank my family and friends for their support and encouragement, which helped me in doing this work.

Nisar Lalani
2005-01-08,
Karlskrona, Sweden

Abstract

Today, testing applications for Internet (web sites and other applications) is being verified using proprietary test solutions. An application is developed and another application is developed to test the first one. 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 thesis 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.

The background topics included are brief introduction of software testing and web testing, short introduction of TTCN language and its version 2 and 3, description of the tool set representing the chosen technologies, conceptual view of how the tools work, a short description of HTTP protocol and description of HTTP adapter (Test Port).

Several benefits and drawbacks were found from all the three technologies but it can be said that at the moment proprietary test solutions (PureTest in this case) is still the best tool to test Internet Application. It scores over other the two technologies (TTCN-2 and TTCN-3) due to reason like flexibility, cost effectiveness, user friendliness, small lead times for competence development etc. TTCN-3 is more of a programming language and is certainly more flexible when compared to TTCN-2. TTCN-3 is still evolving and it can be said that it holds promise. Some of the features are missing which are vital for testing Internet Applications but are better than TTCN-2.

Contents

1 Introduction

This report is submitted in partial fulfilment of the requirements for the Joint European Master's degree in Computer Science given by Karlstad University, Sweden in collaboration with Fachhochschule Wiesbaden, Germany and Fachhochschule Darmstadt, Germany. Test Competence Centre at Ericsson AB, Karlskrona (Sweden), gave the assignment and the work was done at the same location.

1.1 Background

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 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 PureTest from Minq Software AB [9] fits the above criteria more precisely than other tools. It was also decided that the test scenarios written in TTCN notations would concentrate on the above criteria.

1.2 Goal

The goal [22] of this thesis work was to investigate 3 different test technologies/tools to see which one is the best for testing Internet Applications and what are the drawbacks/benefits each technology has.

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 thesis 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.
- Increased knowledge of test and verification of Internet Applications

1.3 Thesis Outline

The remainder of this document is structured in the following manner:

Chapter 2 discusses an overview of Software testing and Web Application testing in particular. This chapter presents the context of Web Application testing and its challenges. Further, chapter 3 presents an overview of TTCN-2 and TTCN-3 notation. Chapter 4 explains the tools for all the three technologies used in this thesis work, i.e. TTCN-2, TTCN-3 and Proprietary Software. A brief report on other available tools is also presented. Chapter 5 presents the concept of Test Port, which is an integral part of TTCN-2 and TTCN-3 tool set. One of the tasks of this thesis work was to make a HTTP test port for TTCN-3 tool set so this chapter also presents an introduction of HTTP protocol and an overview of the test port created to support it. Chapter 6 describes the conceptual view of all the three tools. A brief description of the test cases designed for TTCN-3 environment is also presented. Finally, Chapter 7 concludes the document with an analysis of the three technologies, their advantages and disadvantages. A comparison of all the three tools is also done.

1.4 Target group

This report is mainly aimed at personnel in an organization with software testing as an activity. The reader of this report is supposed to have basic knowledge of elements connected to information technology, Internet, web components and TTCN.

1.5 Limitations

Due to time constraint and other reason like security, not everything in a Web Application could be tested. So this thesis mainly concentrates on some of the issues under static testing and report generation of it.

2 Testing

Software Testing is a set of activities used to verify that a product, service or functionality meets its specification. Testing is never complete and it is almost impossible to perform complete tests due to reasons such as inability to test all the inputs to the program, test all combination of inputs etc. Some of the reasons why testing activities are undertaken by an organization are to manage risks, reduce costs, to better the quality, ensure reliability of the product etc. This chapter describes testing process in brief. It also gives description of Web testing and its challenges.

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. A well-defined and understood way of testing is essential to make the process of testing as effective as possible. In order to produce software products with high quality one has to view the testing process as a planned and systematic way of performing activities. Not only does this process make better tests, but also many testers know from experience that when you think about how to test something, you find faults in whatever it is that you're testing.

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 Mark Fewster and Dorothy Graham [1].

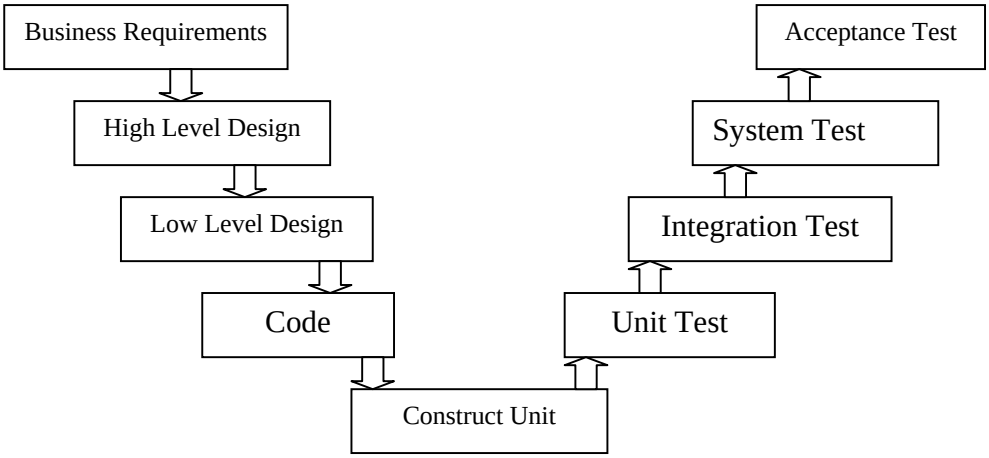


Figure 1 Test Phases in relation with Development phases. V - Model

The above figure shows the typical sequence of development activities on the left-hand (downhill) side and the corresponding sequence of test execution activities on the right-hand (uphill) side. 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.

2.2.1 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.

In theory an independent tester should do this, but in practise the developer usually does it, as they are the only people who understand how a component works. 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. To overcome this, the developer either builds, or uses special software to trick the component into believing it is working in a fully functional system.

2.2.2 Integration Testing

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.

Integration Testing is not focussed on what the components are doing but on how they communicate with each other, as specified in the ‘Low-Level Design’. The ‘Low-Level Design’ defines relationships between components by stating:

- What a component can expect from another component in terms of services.
- How these services will be asked for.
- How they will be given.
- How to handle non-standard conditions, i.e. errors.

2.2.3 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 is not about checking the individual parts of the design, but about checking the system as a whole.

System testing can involve a number of specialist types of test to see if all the functional and non-functional requirements have been met. In addition to functional requirements these may include the following types of testing for the non-functional requirements:

- Performance - Are the performance criteria met?
- Volume - Can large volumes of information be handled?
- Stress - Can peak volumes of information be handled?
- Documentation - Is the documentation usable for the system?
- Robustness - Does the system remain stable under adverse circumstances?

There are many others, the needs for which are dictated by how the system is supposed to perform.

2.2.4 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". It is similar to systems testing in that the whole system is checked but the important difference is the change in focus: Systems Testing checks that the system that was specified has been delivered where as Acceptance Testing checks that the system delivers what was requested.

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 judgement.

2.3 Test Approaches

Test approaches, or test techniques used, will differ throughout the process. Depending on the stage of the test process, type of tests etc. the scope and manner of test will change.

2.3.1 Walkthroughs and inspections

Walkthroughs and *Inspections* are techniques that are used to decrease the risk of future errors to occur. The purpose is to find problems and see what is missing. Walkthrough is a review process in which a designer leads one or more designers, design experts through a segment of design or code, he or she has written. Inspection is a formal evaluation technique involving detailed examination by a person or group other than the author to detect faults and problems.

2.3.2 White-box testing

Testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It focuses on finding faults in the code and architecture of the application.

2.3.3 Black Box Testing

Testing software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a source document, such as a specification or requirements document. Black Box testing as opposed to White Box testing may be done without knowing how things are done, but instead concentrating on what should be done.

2.4 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.

2.4.1 Classification of web sites

Depending on the content and main goal, the website may be classified into a certain type of site. There are a number of different types of sites published on the web. These sites have been categorized by a number of authors. This section presents classifications from two authors who in their way have shown the different angels from which to view/differentiate the sites.

The first classification is based on the different business purposes of a commercial web site. James Ho [5] classifies these purposes of commercial web sites into three categories:

- Promotion of products and services

  Promotion is specific to the products and services that a business offers to customers. Typical examples of this category are product announcements, rates and fare quotes, special offers on products, contests, sweepstakes, giveaways etc.

- Provision of data and information

Provision pertains to the supply of information to gain good will, exposure, credibility, or to expedite communication. Examples of this type category are financial reports, press releases, research data, survey results, benchmark etc.

- Processing of business transactions

  Processing refers to regular business transactions that are beyond the generation of sales leads by promotion. Examples of this category are on-line auctions, interactive brokering, delivery tacking, on-line customer service etc.

Another classification is based on the degree of interactivity the web site offers. Thomas A. Powell [2] classifies web sites into five categories:

- Static Web Sites

  This is the most basic form. It is basically a presentation of HTML-documents. The only interactivity offered is the choice of page by clicking links.

- Static with Form-Based interactivity

  Forms are used for collecting information from the user, including comments or requests for information. The main purpose is document delivery and only limited emphasis on data collection mechanisms.

- Sites with Dynamic Data Access

  These types of web sites are used as front-end for accessing a database. Via the web page, users can search a catalogue or perform queries on the content of a database. The results of the search are displayed as HTML-documents.

- Dynamically Generated Sites

  The wish to provide customized pages for every user creates a need to take a step away from the static web site. The page presented may be static, providing no interactivity, but the way it is created has similarities with the way screens are created in software applications.

- Web-Based Software Applications

  Web sites that are part of the business process often have more in common with other client/server applications than with static web sites. This could be an inventory tracking system or a sales force automation tool.

This classification is derived from the need of methodology during the development of web sites. The classification is useful also for the testing process, not only for the need of methodology but also for how extensive the testing must be. For instance, for a static web site the demands may be,

besides that the information is correct and up-to-date, that the source code is correct and that the load capacity of the server is great enough, i.e. the server can handle a large enough number of visitors at the same time. There is no need to go much deeper in this case. For the other extreme, Web-Based Software Application, the requirements are much greater, where, for instance, security and correctness of transactions is of great importance.

These two classifications are the two major ways of showing distinctions between web sites. Together they provide information about interactivity and purpose which gives us an idea on the site's complexity.

2.4.2 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 authors classifications of web sites was 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, I 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.

Performance testing

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 have 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. Testing web applications for extreme conditions is done by Load and Stress testing. With the help of load testing, it's possible to

determine how effectively a web site or application will accommodate an increasing user/transaction load. By modeling and simulating real-life demands, critical performance and scalability issues can be identified and eliminated. A stress test is designed to determine how heavy a load the Web application can handle. A huge load is generated as quickly as possible while denying it the resources (e.g., RAM, disc etc.) to stress the application to its limit. The idea is to stress a system to the breaking point in order to find bugs that will make that break potentially harmful.

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. For the web, this is often very important due to the users low acceptance level for glitches in the interface or navigation. Due to the nearly complete lack of standards for web layout this area is dependent on actual usage of the site to receive as useful information as possible.

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. Security testing must be done before going "live" and should continue throughout the lifetime of the Web site. This is a highly complex and specialized type of testing.

2.6 Web Testing Challenges

Testing Web Applications presents great challenges due to different level of complexities and diversity of users. An example of complexity is that web application developers often use a great number of different techniques to create the features on their web sites. The mix of techniques used consists of HTML, ASP, JSP, Servlets, Java, JavaScript, ActiveX and others. To test an application containing all this requires a great deal of work and it is a complex process. A scenario where diversity of users makes it difficult to test web site is that many sites today present animations or other graphical effects, which many users consider it as positive. But if you are a visitor searching for specific information, you seldom appreciate waiting time in order to obtain what you seek.

Another problem that always irritates when on the web is broken links. It would be true if we say that there isn't anyone with some web-browsing experience, that hasn't encountered this. It is an always-returning error that will continue to haunt the web for as long as pages are moved or taken off the Internet. These relatively small errors shouldn't be too difficult to remove, and there is therefore no excuse to have broken links on a site for more than a short period of time.

A successful web application can be summarized to as being: usable, secure, scaleable, reliable, maintainable and highly available. If a site does not meet these criteria, then it may run into problems, so all of these factors need to be tested, and that is a complex process.

It can be said that the normal testing challenges exist, such as requirements quality, etc. but the impact is much higher, due to the critical nature of websites, e-commerce websites in particular. Additionally, there are other non-traditional testing efforts that are required for websites, such as performance testing, scalability, and others which add further challenges to a website testing team.

In this thesis work, due to reasons like time constraint, security restrictions etc., effort is made only on the few areas from the ones that are mentioned above. Some of the areas on which I have concentrated are: check of broken links, check of server response time, resource counts, offline browsing, automated testing etc.

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, 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. An example is the testing of OSEK (Appendix D) protocols in the automobile industry.

TTCN defines a test notation, which is used to specify OSI abstract conformance test suites. 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. As the name suggests, test cases are described in the form of behavior trees (Appendix E) and different kinds of tables are used for the

graphical representation of test suites. ASN.1 (Abstract Syntax Notation) was integrated into TTCN as the data notation that is widely used in many communication protocols.

TTCN-2 has two kinds of notations; a human-readable tabular format called TTCN.GR and a machine processable form called TTCN.MP. The MP format is used for the exchange of documents between computer systems.

3.1.1 Basic Notation

A TTCN test suite consists of four 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. Description of these acronyms can be found in Appendix C.

- Constraint Part:

  The constraints part of a TTCN test suite provides the values of the PDUs and ASPs to be sent to or received from the IUT. A PDU constraint is related to a PDU type and describes a constant value or value ranges of the PDU type. Constraints are used in the dynamic part of a test suite.

- The Dynamic Part

  The dynamic part describes the dynamic behavior of the test processes by test cases, test steps and default behaviour 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 behaviour descriptions. A test step can be seen as a procedure definition, which can be called in test cases. A default behaviour description is like a test step, which consists of test situations where the IUT does not behave in an expected manner.

Detailed descriptions of the notation can be found in TTCN-2 standard [3]. Samples of TTCN tables are shown in Appendix B.

3.2 TTCN-3

The new TTCN edition 3 (short: TTCN-3) was launched by ETSI in October 2000. TTCN-3 is intended for various application areas like protocol testing (e.g. mobile and internet protocols), supplementary service testing, module testing, testing of CORBA based platforms, testing of API's etc. [4]. 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.

From syntactical point of view TTCN-3 is very different from earlier versions of the language as defined in ISO/IEC 9646-3 [3]. However, much of the well-proven basic functionality of TTCN has been retained, and in some cases enhanced. A single most visible difference is that the previous versions adopted as the main description notation the Tabular Notation and its textual counterpart a machine-processable language for translation to programs whereas in TTCN-3 the usual programming language-like notation is adopted as the core language with the Tabular Notation and MSC as presentation languages. Comparing it with TTCN-2, TTCN-3 has some major improvements. [4]

- The ability to specify dynamic concurrent testing configurations;
- Operations for synchronous and asynchronous communication;
- The ability to specify encoding information and other attributes (including user extensibility);
- The ability to specify data and signature templates with powerful matching mechanisms;
- Type and value parameterization;
- The assignment and handling of test verdicts;
- Test suite parameterization and test case selection mechanisms;
- Combined use of TTCN-3 with ASN.1 (and potential use with other languages such as IDL);
- Well-defined syntax, interchange format and static semantics;
- Different presentation formats (e.g., tabular and graphical presentation formats);
- A precise execution algorithm (operational semantics).

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 MyModule {
        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. For this, control statements like if-then-else and while loops are supported. They can be used to specify the selection and execution order of individual test cases. The imported module contains the definition for the test case MyTestCase, which is performed in the control part with the execute statement.

The module parameter list defines a set of values that are supplied by the test environment at run-time. During test execution these values are treated as constants and can be accessed within the module from any scope. Groups can be used to structure the declarations of a module to ease the readability.

```
group TestSuiteParameters {
/** the callee user */
        external const charstring CALLEE_USER;
        /** the caller user */
        external const charstring CALLER_USER;
        …
} // end group TestSuiteParameters
```

As in other programming languages, constants are used for declarations of static data.

```
group ConstDefinitions {
        const charstring Version := "MyApp/2.0";
        /** @ (at) sign */
        const charstring AT := "@";
        …
} // end group ConstDefinitions
```

Types of protocol messages are defined as structured types, which can be constructed as record, set, enumerated types, etc. from basic types or other structured types. TTCN-3 supports a number of predefined basic types. They include typical programming basic types such as integer, boolean and string types, as well as testing-specific types such as verdict type, port and component type. The port and component types are used to define the configuration of a test system. An example of a message type with optional information elements is given below.

```
type record Request {
        RequestLine requestLine,
        ReqMessageHeader reqMessageHeader optional,
        charstring crlf,
        MessageBody messageBody optional
}
```

Test data, which is to be sent or received over ports, is defined as templates. Templates support matching mechanism for the received data. In addition, parameterization of templates is used for flexibility in using templates in a current testing context. The template MyRequest_1 given below is of type Request. It defines completely the data to be sent. The messageBody field is omitted.

```
template Request MyRequest_1 := {
        requestLine := Request_Line_1,
        reqMessageHeader := Req_Message_Header_1,
        crlf := CRLF,
        messageBody := omit
}
```

Test cases describe the test behavior. Test cases are executed by a main test component, which is automatically created when a test case is started. It is also possible to create and terminate parallel test components during the execution of a test case. The test components (both the main test component and the parallel test components) are connected via well-defined communication interfaces, so-called ports, to the system under test or to other test components.

A test configuration is a set of test components. The set of ports (interfaces), which are belonging to a component are defined in the respective component type. A component type can also define local variables and timers, which are instantiated whenever a component of that type is created.

```
group TestConfigurationDefinition {
        /** communication port type definitions */
        type port MyPortType message {
                inout Request, Response;
        }
        /** component type definitions */
        type component MyTestComponent {
                timer  MyTimer1 := 0.5; // MyTimer1 = 500 ms
                timer MyTimer2 := 4.0; // MyTimer2 = 4s
                port MyPortType My_PCO
        }
} // end group ConfigurationDefinition
```

The MyPortType is a message-based port. It is a bi-directional port and supports Request and Response messages in both the directions. The component type MyTestComponent defines two local timers MyTimer1 and MyTimer2 (the unit of timer values are in seconds) and one port of type MyPortType.

A variety of test relevant behavior can be expressed in a test case such as the alternative communication event reception, their interleaving and default behavior e.g., unexpected reactions for the tested systems. The test case MyTestCase shown below has a very simple behavior. At first a default is activated for handling unexpected or irrelevant responses during test execution. Next, MyRequest_1 is sent, the timer MyTimer1 is started, which avoids never-ending waiting for the expected response MyResponse_1. If this is received successfully (i.e. the response of the system under test has matched the template), the timer is cancelled. A pass verdict is then assigned and the test terminates.

```
testcase MyTestCase() runs on MyTestComponent {
        activate (Default_1); // Default activation
        My_PCO.send(MyRequest_1);
        MyTimer1.start;
        // the CSeq value has been not understood
        My_PCO.receive(MyResponse_1);
        MyTimer1.cancel;
        setverdict(pass);
        stop;
} // end testcase
```

In TTCN-3 defaults are used to handle communication events, which are unexpected or irrelevant. Defaults in TTCN-3 are handled as macro expansion: when activated, they are expanded automatically as last alternatives to receiving events. Several defaults can be activated and/or deactivated.

4 Tool Description

This chapter presents the tools (TTCN-2, TTCN-3 & PureTest) 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 PureTest. Finally, a description of other tools is given.

4.1 TTCN-2 Tool

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.

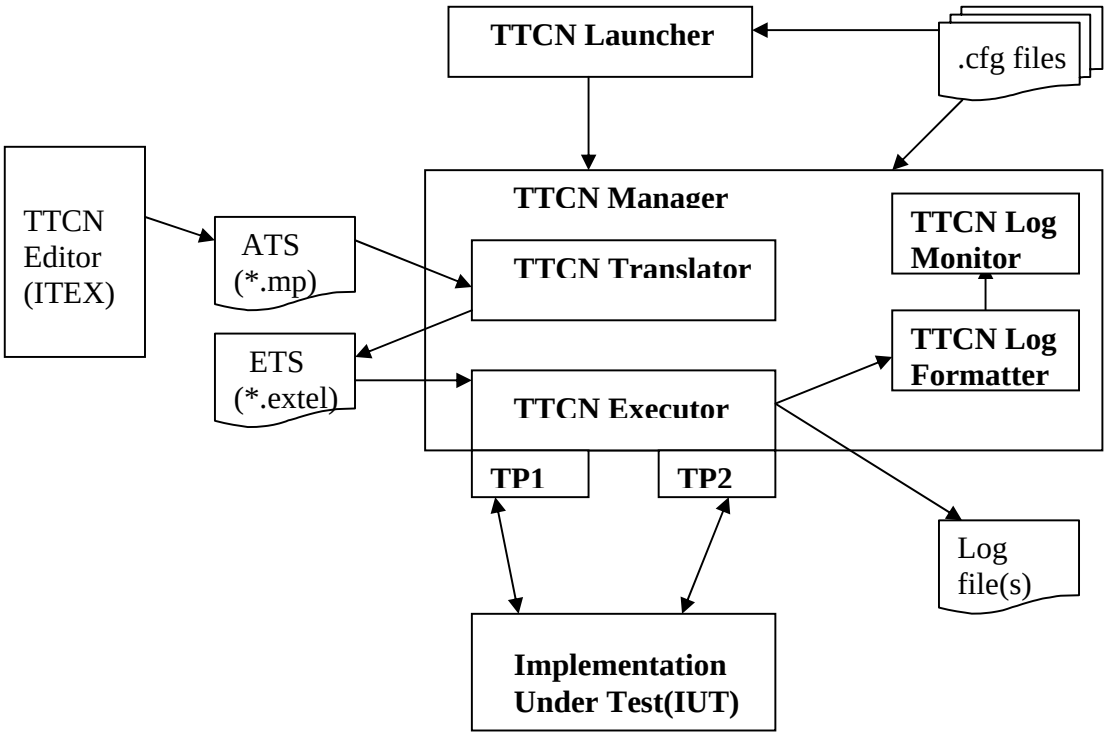


Figure 2 TTCN-2 Test System

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 (from Telelogic[6]) which is most commonly used. Test suites

written in ITEX are then saved in .mp format and are known as Abstract Test Suites (ATS). It is possible to use this file in all tools that follows the TTCN standard.

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 Manager provides interactive execution meaning values can be assigned during runtime rather than having to specify them in test cases. A Configuration file, which provides the setup parameters for the test execution, is loaded into the TTCN Manager.

TTCN Translator

It is a compiler which converts the TTCN machine processable format (.mp) into a internal format Executable Test Language(ExTeL). Translator can take multiple TTCN test suites and generates ExTeL files for each one of them. The ExTeL-file is also called ETS, Executable Test Suite. The ETS format is not standardized, it is an executable version of the ATS, and different tools have different formats for their ETSs.

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. Chapter 5 Test Port, presents more detailed description on test ports.

The result from the execution is presented in a window and is also saved as a text-file. It consists of listings about which signals and what data were sent and received and the eventual verdicts. No graphical visualization of the result exists.

4.2 TTCN-3 Tool

Titan, an internally developed tool by Ericsson, is a Ericsson wide official tool set for TTCN-3. Their are various other TTCN-3 tool set vendors such as Telelogic[6], Testing Tech [7], Danet [8] etc. Since Titan is used during this thesis work, I have chosen to describe it here.

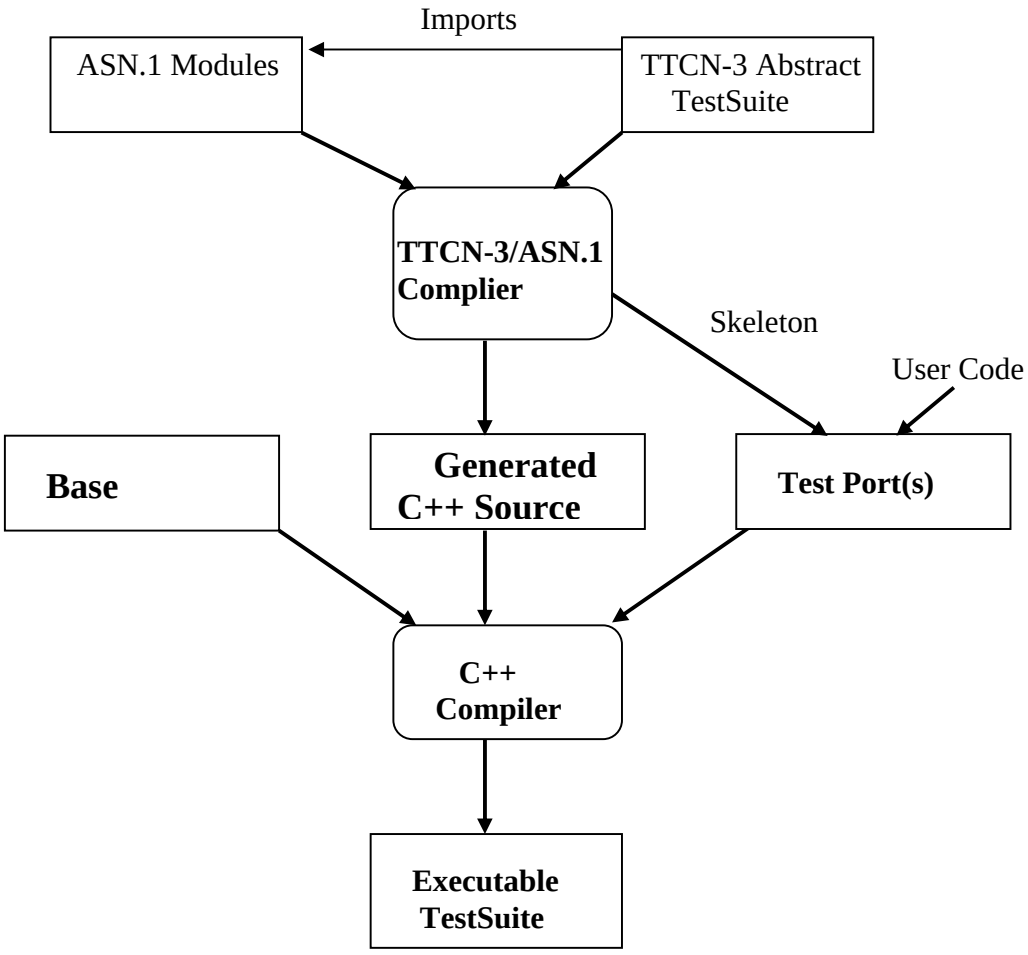


Figure 3 Structure of TTCN-3 Executor

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

The generated C++ code and Test Ports have to be compiled to binary object code and linked together with the Base Library using a traditional C++ compiler.

4.3 Pure Test

The Pure Performance suite by Minq Software AB [9] offers integrated solutions for test and verification of Web applications. The suite includes:

- **PureTest**

  It is a single user application which is primarily used to setup scenarios of tasks, execute and debug them. PureTest includes the HTTP Recorder and Web Crawler which makes it useful for generic verification of HTTP requests and web content checking.

- **PureLoad**

  Pure Load is a complete load testing solution. It is a generic and distributed load testing product with support for testing applications over a wide range of protocols such as JDBC (RDBMS), FTP, LDAP, DNS, JMS, MAIL, etc. It also includes extensive support for testing of HTTP based applications. The HTTP Recorder and Web Crawler utilities are used to capture HTTP requests into PureLoad scenarios. Custom scenarios can optionally be created in Java using the Task API.

- **PureAgent**

  This is a performance-monitoring product which periodically executes a scenario of tasks in order to report the performance as seen from end user perspective. The PureTest product is delivered with PureAgent and is used to design and setup the actual scenarios. The PureAgent product is controlled using a web-based interface. Reporting can be integrated with third-party monitoring applications such as SysLoad, Patrol or Tivoli.

Pure Test is free of cost while PureLoad and PureAgent are commercial tools offering free evaluation for a brief duration. I have chosen PureTest for my thesis work because it is free to use and offers some good features. Other tools like Rational Robot, Astra Quick Test etc are not free.

Although they offer evaluation versions, it was noticed that the period of evaluation was very short. In this section a brief description of the PureTest tool is presented.

4.3.1 Tool Overview

Testing web based application using the HTTP protocol has become crucial since many applications today uses a web browser as its only interface. PureTest offers extensive support for testing such applications. Testing using PureTest is done by designing and parameterizing scenarios, executing and evaluating the results. It is a single user application. It includes two utility tools; HTTP Recorder and Web Crawler, which makes it useful for verification of HTTP requests and web content checking.

The PureTest graphical user interface is used to design scenarios and the main part consists of the Scenario Editor, i.e. the editor used to manage scenarios, task parameters and parameter generators as well as test and debug created scenarios. Scenarios of tasks (functions) is the generic foundation throughout all applications in the Pure Performance suite. A task is the actual function that shall be performed, i.e. get a web page, insert data into a database, etc. A series of tasks are defined in a scenario. An extensive set of features is available to make tasks modular and dynamic. Details about scenario editor can be found in the Scenario Editor User's Guide[10].

4.3.2 Utility Tools of PureTest

There are two tools available in Pure Test which assist in creating accurate scenarios:

- HTTP Recorder

  It records all requests and parameters between a web browser and the web server. The result is organized in scenarios containing task sequences (that represents a web page) and individual tasks that represent each HTTP request.

- Web Crawler

  It crawls through a static web site and presents information about resources, statistics, broken links, etc.

These utilities are also part of the PureLoad application. Below is a brief introduction of these tools.

HTTP Recorder

The following figure illustrates the runtime environment for the HTTP Recorder

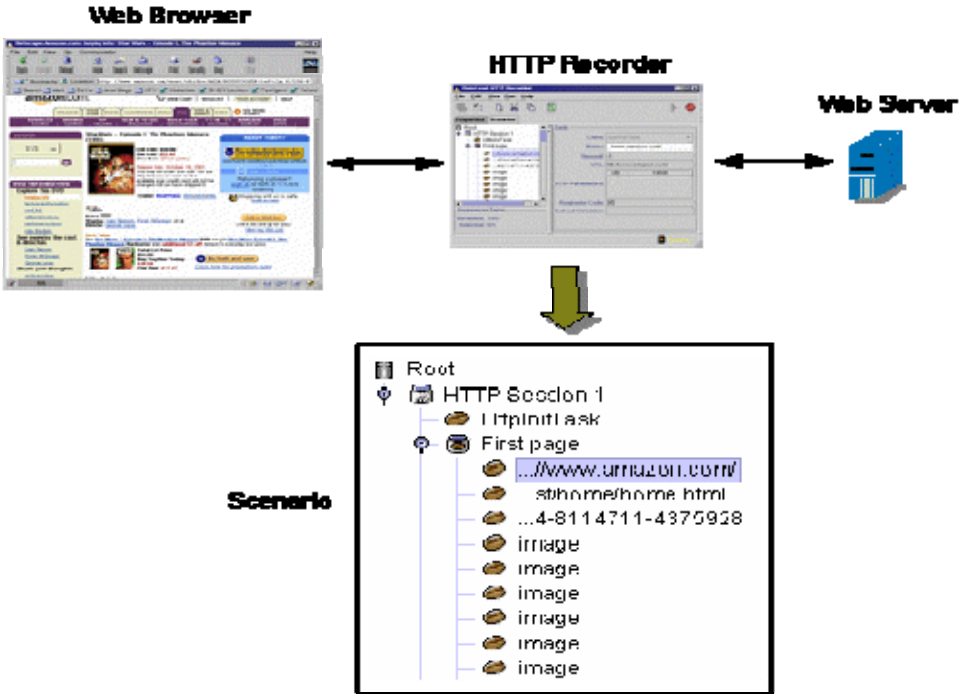


Figure 4: Architecture of HTTP Recorder

The figure above shows how the HTTP Recorder is attached to a web browser as a proxy server. All requests that are passed from the web browser go through the HTTP recorder to the web server. The response from the web server is passed back through the HTTP recorder and then displayed in the web browser. HTTP recorder captures all information that is transferred on the wire and creates a scenario with tasks. The recorder makes sure the appropriate HTTP tasks are used for the HTTP GET and PUT requests. All parameters supplied by the browser are also recorded into the tasks.

Web Crawler

The web crawler is a utility that is used to create scenarios for static web content, i.e. HTML, GIF, JPG files. The crawler uses the content of a specified web page to find out all related resources (links). It then traverses the information until no more references can be found.

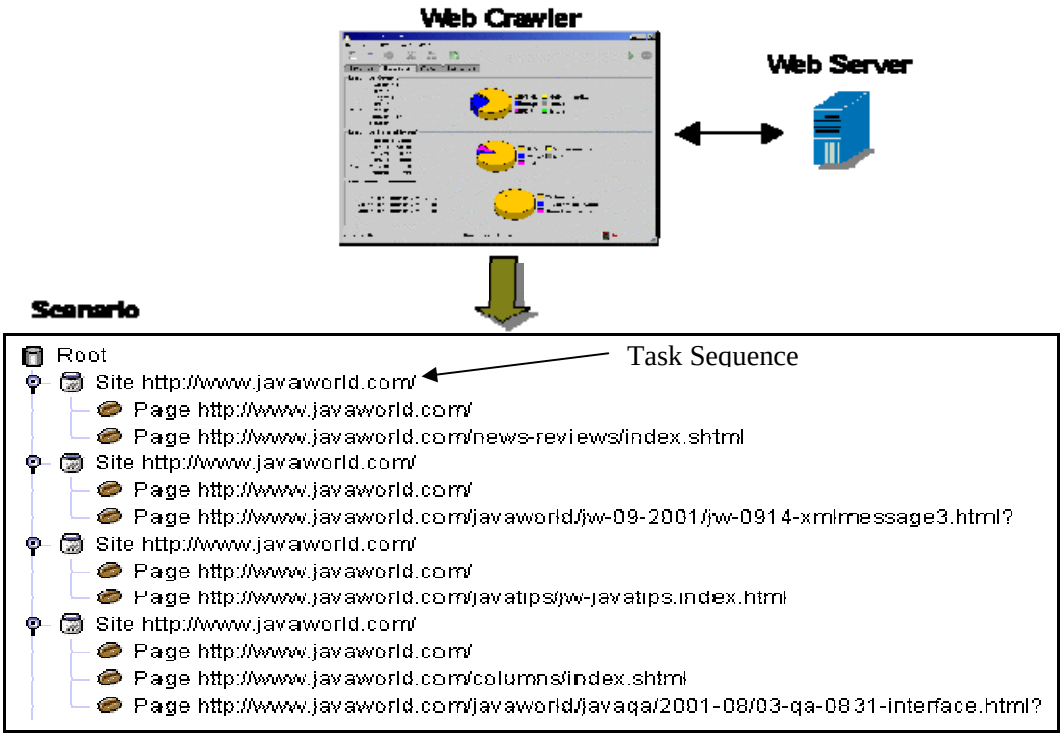


Figure 5: Architecture of Web Crawler

The crawler also presents various statistics (Web Crawler result in the above figure) that can be used to determine the size of all resources, number of resources per type, etc. It also reports errors that are found when crawling the web site.

When a website has been crawled, scenarios can be generated from the result set. Above figure gives an example. The crawler generates scenarios based on the selected resources. A HTML page will be created as a Task Sequence in the generated scenario while other resources will be Tasks. A task sequence is a container in which tasks and other task sequences can be grouped. A Task is a Java class implementation that encapsulates an operation that will be executed in the run time environment. PureTest comes with the predefined tasks that support testing of the most commonly used protocols and network applications. All the common methods of HTTP protocols like GET, PUT, DELETE are encapsulated in PureTest's tasks. The generated scenario information can be saved to a PLC (PureLoad Configuration File) and later be opened in PureLoad or PureTest in order to use it in a load test configuration.

More details about PureTest and its utilities can be obtained from online documentation[11]

4.4 Other Tools

Before the start of the thesis, certain criteria (Chapter 1.1) where set for the validation of Internet Applications and depending on these criteria, a tool had to be selected. There are several software tools available in the market which are in the business of Web testing. Some are completely free, some are highly expensive but offer free evaluation licences, some are not so expensive but do

not offer free evaluation, some are highly sophisticated, advanced and evolved, some are immature, some are designed specifically for single purpose for example, Link Checkers, HTML validators, some offer the whole range of services. During the selection process, lot of tools where considered but the finally PureTest from Minq Software AB [9] was selected as it met the requirements to the closest. This section gives a brief overview and feature list of some of the tools, which were considered.

Astra QuickTest

Astra QuickTest together with a suite of other testing tools comes from Mercury Interactive [12]. It is a testing tool that creates scripts for static and dynamic websites to validate all steps of a process. It also checks for number of links, images, text area elements, select elements, forms, frames, input elements etc and produces the results in easy-to-read graphical and tabular reports. Script development in Astra Quick Test comes with a recording function, which records the user's actions and progress through the site, and saves the script as an individual file. The user is then able to edit and enhance the generated scripts with a scripting tool. These scripts can also handle multiple sets of data for running simultaneous tests (different sets of login data, multiple values for inputs, etc.). It verifies most Web page components for quick regression testing. It also performs database verification to ensure transaction accuracy.

Astra Quick Test supports many complex web environments, including: Java (JDK including JFC, AWT and Symantec's Visual Cafe), Real (record/playback for streaming media), Flash (record/playback for Flash animation), and ActiveX Controls (enables testing of interactive content on the Web).

It supports multiple versions of IE and NS. But it runs only on Windows as a platform, and doesn't support Opera, AOL, etc.

Astra Quick Test though very sophisticated, is a very expensive tool. It costs around $4000.

Link Scan

LinkScan is an industrial-strength link checking and website management tool. It is cross platform compatible. It can scan local and remote files. It can check links embedded in drop down menus, ASP, CFM, JSP, JavaScript, Flash and DHTML. There are many more features to this product but it appears to be a Link checker with all the bells and whistles. It also contains a feature called LinkScan Recorder that works with IE.

LinkScan is developed by Electronic Software Publishing Corporation [16] and costs around $750-$4000 depending on the usage and number of users.

WebKing

WebKing is a Web verification tool that automates static analysis, functional testing, and load testing. WebKing's wizards and automated technologies helps to verify application functionality and reliability without writing a single script, plus its flexibility allows to automatically verify even the most specialized requirements.

Features:

- Verifies functionality, performance, and accessibility with one-click
- Automates static analysis, functional/regression testing, and load testing
- Designs functional tests for application's most popular paths
- Designs realistic load tests by analysing server log files
- Dynamically populates forms with data source values
- Verifies custom group guidelines and project requirements
- Verifies Section 508 accessibility compliance
- Integrates with WebSphere Studio Application Developer and Eclipse

WebKing is developed by ParaSoft Techonologies [17] and costs around $3500

Rational Robot

IBM Rational Robot [18] automates regression, functional and configuration testing for e-commerce, client/server and ERP applications. Rational Robot's integrated component, SiteCheck is a powerful tool for testing the structural integrity of intranet or World Wide Web site. It is designed to view, track, and maintain rapidly changing web sites.

SiteCheck provides numerous features including:

- Support for Netscape Navigator, Microsoft Internet Explorer and many other browsers.
- Automatic tracking and updating of broken links, permanently moved or orphan pages, and pages that are slow to download.
- Automatic updating of broken links, permanently moved or orphan pages, and pages that are slow to download.
- Identification and verification of Web pages with active content including ActiveX and Java.
- Support for Secure Sockets layer (SSL)
- Support for proxy servers,
- Virus checking with McAfee VirusScan 2.0 or later.
- A fully functional, color-coded HTML editor and source code browser.
- Informative reports

Doctor HTML

This tool is not free and does not offer any free evaluation period. However, the feature list given below suggests that this tool fits the criteria set for this thesis work. The main reason of not selecting this tool was due to its cost, which is $350 for a yearly licence. The following are the features available in the current release (v6.1) of Doctor HTML. [13]

- It has an Easy-to-use Web Interface for report configuration.
- The reports generated are informative and nicely formatted with hyperlinks to additional information.

- Fifteen report options
- It checks the document for spelling errors (provides suggestions for potentially misspelled words)
- Perform an analysis of the images (size, number of colors, etc.)
- Tests the document structure and flags invalid HTML
- Examine image syntax (flag missing, but recommended elements)
- Test browser support of tags used
- Examine fonts in page to check for cross-platform support
- Examine table structure
- Verify that all hyperlinks are valid
- Examine form structure
- Show command hierarchy
- Produce a "squished" version of the page
- Produce a "formatted" version of the page
- Expand frameset for further testing
- Check Meta tags for search engine relevance
- Show cookies set by the page
- Ability to test Web pages that are password-protected (HTTP Authentication)
- Site Doctor: Diagnoses the entire Web site, or a subset of pages. It presents a site map showing page connectivity and individual Doctor HTML reports for each page.
- Ability to write custom modules for own tests
- Customisable spelling word lists

Open Source Tools

HT://Check

HT://Check [19] is a console application based on ht://Dig, a free search engine. It allows web developers to not only find and report on broken links, but also other information from HTML documents. It stores its data in a mySQL database that can also be used for custom queries.

HT://Check is made up of two logical parts: a "spider" which starts checking URLs from a specific one or from a list of them; and an "analyser" which takes the results of the first part and shows summaries (this part can be done via console or by using the PHP interface through a web server).

htcheck (the console application) gives the summary of broken links, broken anchors, servers seen, content-types encountered. The PHP interface performs:

- Queries regarding URLs, by choosing many criterias such as pattern matching, status code, content-type, size.
- Queries regarding links, with pattern matching on both source and destination URLs (also with regular expressions), the results (broken, ok, anchor not found, redirected) and their type (normal, direct, redirected).
- Info regarding specific URLs (outgoing and incoming links, last modify datetime, etc.

- Info regarding specific links (broken or ok) and the HTML instruction that issued it
- Statistics on documents retrieved

InSite

InSite [20] is a free Web site management tool that is written in Perl. It offers a lot of information in its reports, including link verification, link type reporting, e-mail alerts on user-defined critical pages, flagging overstuffed pages, and scanning pages for user-defined test strings. It also reports on what file types are being used, and the types of link errors received on links to remote sites.

Below is the feature list as described in [20]:

- fast local link verification: 76,000 documents containing 1.5 million links were verified in 2 hours on a PII 300 with 512 MB RAM running RedHat Linux 6.2 with all web files mounted via NFS over 100 Mbps Ethernet (the 1.3 million local links took 95 minutes to verify -- 230 links per second)
- remote link verification and classification by error type
- lists all mailto links' email addresses and frequency of use
- provides email alerts for any errors found on user- defined "critical pages"
- calculates download sizes for critical pages
- scans pages for the presence or absence of red flags, user-defined text strings (defined using powerful Perl regular expressions)
- runs CGI programs and validates output
- full report of all broken links, including the document in which the link was found and the document to which the link was pointing.
- provides summary statistics:
    - number of links per document by link type (A, IMG, APPLET, etc.) and number and percentage of each that are broken
    - number of documents by file extension
    - page size breakdown (how many documents are between 1K and 5K? how many are between 50K and 100K?)
- builds a simple text database of all parsed documents and the links they contain so you can quickly find all links to a particular document
- can be configured to ignore subsets of the site and to ignore specified remote links
- can validate user pages (translates URLs of the form "/~username")
- understands Apache-style SSIs

There are various other open source tools for function testing(WebTst, Enterprise Web Test etc.), performance testing(Diesel Test, Hammerhead , JSpider etc), Link Checking (Jenu, Link Checker etc.) etc. More information about them and various other tools can be found in Open Source Testing website [21].

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.

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. Both TTCN-2 and TTCN-3 achieve this goal by including a possiblity of adding new interface adapters to the system without having any need to change the core executor.

These interface adapters are called test ports. In the case of TTCN-2 tool set, these are dynamically linked to the TCE when the test session is started. In TTCN-3 tool set, it is linked together with the executable test program. For each type of interface where TTCN usage is required, there has to be a test port designed. The main task of the test port is to manage the communication between the test system and the Implementation Under Test (IUT). The Test Port offers services to the Test Execution called Abstract Service Primitive (ASP). These ASP's also have to be declared in the test suite in the way the Test Port offers them. The Test Port usually contains a number of protocol layers depending on the System Under Test's interface.

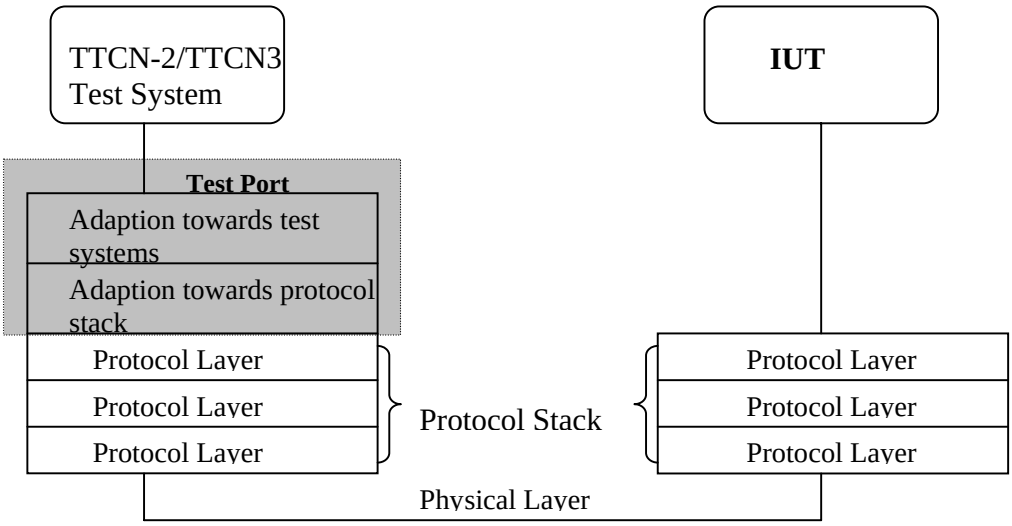


Figure 6 , Conceptual view of Test Port

A test port consists of two parts. The first part is an adaptation to the Tools(SCS Tools/Titan Tools) which is used when it's linked into the Tools. This adaptation is auto-generated when using the test port generator tool. The next part of the test port is the adaptation to the protocol stack, which must be designed to handle the current protocol. The protocol stack is often a system

consisting of software and sometimes also hardware, which implements the protocol stack, offering a number of ASP's to be used by the executed test suites.

Each PCO in TTCN-2 (ports in TTCN-3) used within a Test Suite will have a test port appointed when the Test Suite is executed.

Based on this concept, the HTTP test port for TTCN-3 has been designed. This is presented in the next section. This test port is based on the similar test port made by Ericsson AB which is a part of TTCN-2 environment.

5.2 HTTP Basics

To test a web application, basic understanding of HTTP is needed. There is also a need to know how the application which is being tested, uses HTTP.

This section describes some HTTP concepts that are important when testing a web application. A more detailed description of HTTP is presented in HTTP standard [15].

5.2.1 The HTTP Request/Response Model

HTTP is based on a simple, Request/Response model. A client sends a request for a resource to a server, and the server sends back a response with the requested resource (or a response with an error message if the resource is not available). A resource can be a static resource (such as a HTML file) or a dynamically generated resource.

HTTP Request

An HTTP request message consists of three parts: a request line, request headers and sometimes a request body. The request line starts with the request method name, followed by a resource identifier and the protocol version:

GET /index.html HTTP/1.0

The most commonly used methods are GET and POST. A GET request, as the name suggests, is used to get a resource on the web server. This is the default method, so if we type a URL in the browser or click on a link, the browser or client sends the request to the server as a GET request.

The request headers provide additional information the server might need, such as what browser that is used (*User-Agent*) and headers to provide information about languages and what file formats the browser accepts.

As an example of an HTTP GET request, let us say that we want to retrieve the file specified by the URL http://www.dummyhost.com/path/file.html. The browser first opens a socket to the host www.dummyhost.com on port 80 (default port is always 80 for HTTP requests if nothing else is specified in the URL, e.g. http://www.somehost.com:90). Then, the browser sends something like the following through the socket:
GET /index.html HTTP/1.1

Host: www.myhost.com
User-Agent: HTTPTool/1.1
[blank line here]

HTTP Response

The response message is similar to the request message. It consists of: a *status line*, *response headers* and possibly a *response body*. The status line start with the *protocol version*, followed by a *result code* and a short description of the result:

HTTP/1.1 200 OK

An example of the response from the request described in the previous section could look like this:

HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
<h1>Some Home Page</h1>
.
.
.
</body>
</html>

Request Parameters and Request Methods

Besides the URL and headers, a request message can contain additional information as HTTP parameters. Parameters can be sent either by appending them to the URL in the form of a query string or sent as part of the request body.

An example of a GET request with parameters sent as part of a URL with a query string:
GET /search?name=Tommy+Nilsson&city=karlskrona HTTP/1.1

The query string starts with a question mark (?) and consists of name=value pairs separated by ampersands (&). These names must be encoded to avoid confusion with characters used to separate name=value pairs (URL encoded).

As described earlier, GET is the most commonly used request method. The next common method is POST. A POST request is used to send data to the server to be processed in some way, one example is a *CGI* (Common Gateway Interface) program.

The request parameter is the most obvious difference between the GET and POST methods. A GET request always uses a query string to send parameter values, while a POST request always sends them as part of the body. There are usually also extra headers to describe the message body, like *Content-Type* and *Content-Length*.

Besides GET and POST, HTTP has a few more methods which are not so common. Some of them are HEAD, PUT, etc.

State Management

HTTP is as stateless protocol. When the server sends back a response to a request, it forgets all about the transaction. If a user sends a new request, the server has no way of knowing if it is related to the previous request. This is a problem for a web application where a number of requests may be needed to complete a transaction. The shopping cart is a classic example - a client can put items in his virtual cart, accumulating them until he checks out, several page requests later.

Over the years basically two ways to overcome HTTP's stateless nature has been used. The server can either return the complete state with each response and let the browser send it back as part of the next request, or it can save the state on the server and send back only an identifier that the browser returns with the next request. The identifier is then used to locate the state information saved on the server.

In both cases, the information can be sent to the browser in one of three ways:

- As a cookie
- Embedded as hidden fields in an HTML form
- Encoded in the URLs in the response body, typically as links to other pages (this is known as URL rewriting)

Cookies

A cookie is a name/value pair the server passes to the browser in a response header. The browser stores the cookie and sends it with the next request to the same server in the request header.
Example of a cookie:
Cookie: JServSessionIdroot=w40ac50i78

Using cookies is the easiest way to deal with the state issue, but cookies are not supported by all types of browsers. In addition, a user may disable cookies in a browser because of privacy concerns. Hence, most web applications do not rely on cookies alone.

Hidden Form Fields

If hidden fields in an HTML form are used to send the state information to the browser, the browser returns the information to the server as regular HTTP parameters when the form is submitted. Example of a hidden field:
<INPUT TYPE=HIDDEN NAME="Id" VALUE="87141">

URL Rewriting

This is simply a reference to other pages, where the URL includes additionally state information. Example of a URL with a query string:
http://jsecom8a.sun.com/servlet/EComActionServlet?StoreId=8&Part=12

HTTP has various other features like authentication management, various response codes etc. Detail information about HTTP can be found in [15].

5.3 HTTP Test Port

This section presents a brief description of the HTTP test adaptor, which was developed for the TTCN-3 tool set during this thesis work.

The primary function of the testport is to work as a protocol interface between the generic, protocol independent Titan's Run Time Environment (RTE). The configuration of the HTTP test port is performed by editing the run-time configuration file. The behavior of the executable test program is also described in the runtime configuration file. It is a simple text file, which contains various sections after each other.

The work of the testport can be divided into three phases: the setup phase, the traffic phase and finally the closing down phase. The setup phase, is characterized by first fetching the content of the configuration file and then a socket connection is created between the testport and the Implementation Under Test (IUT).

In the traffic phase, messages flow to/from the testport in both directions, i.e. towards RTE and towards IUT. The way the messages are being transferred differ for the two interfaces of the testport. In the upper interface, towards the RTE, the messages are carried by function calls as the software of the testport is linked with the RTE software. In the lower interface, towards the IUT, TCP socket connections are used as a bearer of the messages.

The closing down phase ends the work of the testport. The order to disconnect the testport from the server comes in a function call from the RTE. The testport responds by closing down the connection to the IUT.

6 Test Scenarios

This chapter presents the test scenarios when using TTCN-2,TTCN-3 and Pure Test. All three technologies basically use the HTTP Request/Response model. The test cases made during the thesis work follow the same logic in both TTCN-2 and TTCN-3 and therefore in this chapter only TTCN-3 test cases are described. The first section describes TTCN-3 test scenarios followed by the description of TTCN-2 and Pure Test.

6.1 TTCN-3

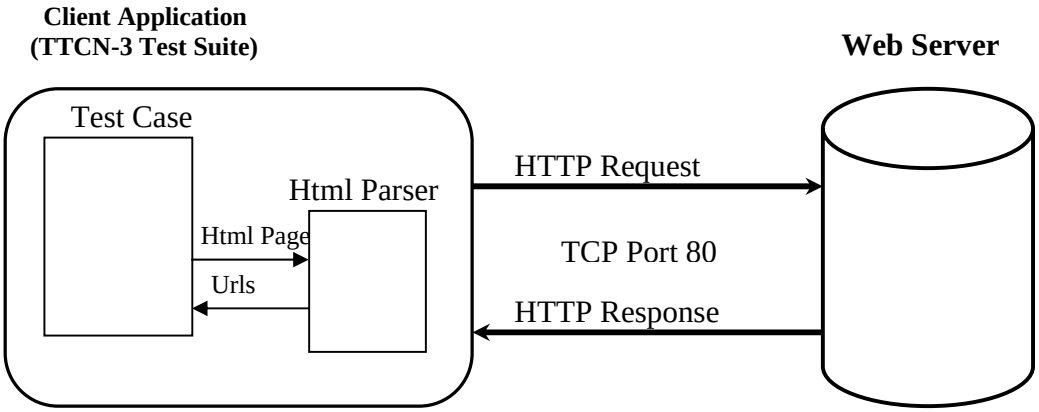


Figure 7: Test Scenario using TTCN-3

The figure above shows the test scenario when using TTCN-3 system. The client application sends request to the web server who sends the response back to the client. The client application is a TTCN-3 test suite which contains the test cases and an external function which does the Html parsing. In order to send this requests, a client program uses the HTTP Test port which facilitates the message exchange. The test port establishes a TCP connection and sends the messages to and receives messages from the WebServer. The text below explains the request/response message format during the execution of tests.

6.1.1 HTTP Request/Response Format

An HTTP request is typically initiated from a client application. The different parameters are packed into a request and sent to a Web Server over a TCP/IP connection.The end result is an opening of a TCP/IP connection to the Web Server on port 80, and transmission of the HTTP request. This connection setup to the Web Server is done by the HTTP adapter. The syntax of the messages are are already described in section 5.2.

Since the test cases are using a GET method to request information, the body content is left empty. To send this request, a client program uses a HTTP test port which opens a TCP/IP connection and sends the request to the server.

An example request is presented below, using the following elements:

IP address=10.201.2.31
method = GET
URI = http://esekant027.epk.ericsson.se/042/
Major = 1
Minor = 1

GET http://esekant027.epk.ericsson.se/042/ HTTP/1.1 Host:10.201.2.31<CRLF>

The web server will respond on the same TCP/IP connection from which the request was received.

Example:

HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
<h1>Welcome To My Home Page</h1>
.
.
</body>
</html>

If the status is OK, the request has been answered and the body content can be used. Any other status code means an error occurred and the status text contains a detailed description of the error. The Body contains the result the resource which was asked. As the HTTP Test Port is designed in such a way that Request/Response could only handle character strings, other resources as for example gif, pdf, powerpoint etc are not possible to handle. So, for every resource which is of type other than 'text/html', the Body Content is ignored by HTTP Test Port.

6.1.3 Example Test Case

In this section, a test case is presented. It establishes a connection and sends a request to the web server, the server replies back with its response message depending on which the result is derived. This test case fetches just one html page. In the thesis work, this concept is applied for the testing of the whole website.

```
template Message get_ttcn_site( template Request req ) := {
        request := req
}
//request url
template Request requestUrl (HeaderLines headerFields) := {
        method := "GET",
```

```
        uri    := "http://esekant027.epk.ericsson.se/019/",
        major  := 1,
        minor  := 1,
        header := headerFields,
        body   := omit
}
template Message rec_ttcn_site(template Response rec) := {
        response := rec
}
//Response
template Response receivedOk := {
        major := 1,
        minor := 1,
        statusCode := 200,
        statusText := "OK",
        header := ?,
        body  := *
}
//Test  Case
testcase simple( ) runs on client
{
1       map(mtc:http, system:http);
2
3       var charstring host := "http://esekant027.epk.ericsson.se";
4       var integer portNo := 80;
5       var HeaderLines headerFields := {"Host: epkws1238", "Connection: close"};
6       http.send(conn(host,portNo));
7       http.send(get_ttcn_site(requestUrl(headerFields)));
8       alt{
9       [ ] http.receive(Message:rec_ttcn_site(receivedOk)){
10              setverdict(pass);
11      }
12      [ ] http.receive{
13              setverdict(fail);
14      }
15      };
16      unmap(mtc:http, system:http);
}
```

Without going into very much details of the syntax of the above test case, let us look briefly at the code line by line.

Line 1 : Mapping is done between the main test component with the system component.

Line 3 to 5 : Variables containing host, portnumber and header fields are created.

Line 6 : A connect(conn) request is send to the Web Server with the host and port number. If the connection is successful, the test case proceeds further otherwise a run time error is generated.

Line 7 : A request is send to the Web Server, for the resource(specified in the template requestUrl).

Line 8 : Starts the alternative section. A snapshot is taken in the TTCN-3 run time environment (Titan) when something is received on its ports. With this snapshot, the run-time environment get the information form which it can derive whether a test case has passed or failed.

Line 9-11 : If the message received matches the template (*receiveOk*), the verdict is set to '**pass'** meaning the test case has succeseded

Line 12-14 : If any message other than *receiveOk* is received, the verdict is set to '**fail'** meaning the test case has failed.

Line 15 : End of the alternative section.

Line 16 : Proper close down of the components.

The above test case is used to fetch one resource from the WebServer. This method is quite good when the requirement is to check very few specific resources in a website. However, when it is required to check the whole website or a big part of a web site, the above procedure to check the resources one by one becomes a big challenge and waste of valueble time and resources. To tackle this, the test cases can be made more intelligent so that when a resource is fetched from the website, its body content can be checked for the possiblity of other resources in it. If there are any resources, test can be performed for them and process repeates itselves till all the resources are tested.

Here's a pseudocode summary of the algorithm on which the test cases were made. Please see Appendix-A for complete source code of the test suite. :

```
BEGIN
Get the test suite parameters : The starting URL, the Host Name, and the starting directory.
Establish a connection with the WebServer.
Add the starting URL to the currently empty list of URLs to search.
Fetch the resource specified in the URL from the webserver
IF fetching was a success THEN
  Find out the content type and content length
  Print the test case report
  IF its an HTML resource THEN
    Call: HtmlParser(body)
    IF HtmlParser returned any resource list THEN
      Store the resources into the list of urls to search
        FOR each resource in the list {
          Check if it was processed earlier
          IF it was not processed THEN
            fetch the resource from the webserver by calling a LoopFunction.
```

```
            }
        }
    }
  }
  update the statistics(body size,resource count etc.)
}
IF fecthing was a failure THEN
Print the error report
}
Print the Final Report
Print the total structure
END

BEGIN LoopFunction
Add the URL to the list of URLs to search.
Fetch the resource specified in the URL from the webserver
IF fetching was a success THEN
  Find out the content type and content length
  Print the test case report
  IF resource is a HTML page THEN
    Call: HtmlParser(body)
    IF HtmlParser returned any resource list THEN
      Store the resources into the list of urls to search
        FOR each resource in the list {
          Check if it was processed earlier
          IF it was not processed THEN
            fetch the resource from the webserver by calling a LoopFunction.
          }
        }
    }
  }
  update the statistics(resource type,body size,resource count etc.)
}
IF fetching was a failure THEN
Print the error report
}
END
```

Complete test case is given in Appendix F.

6.2 TTCN-2

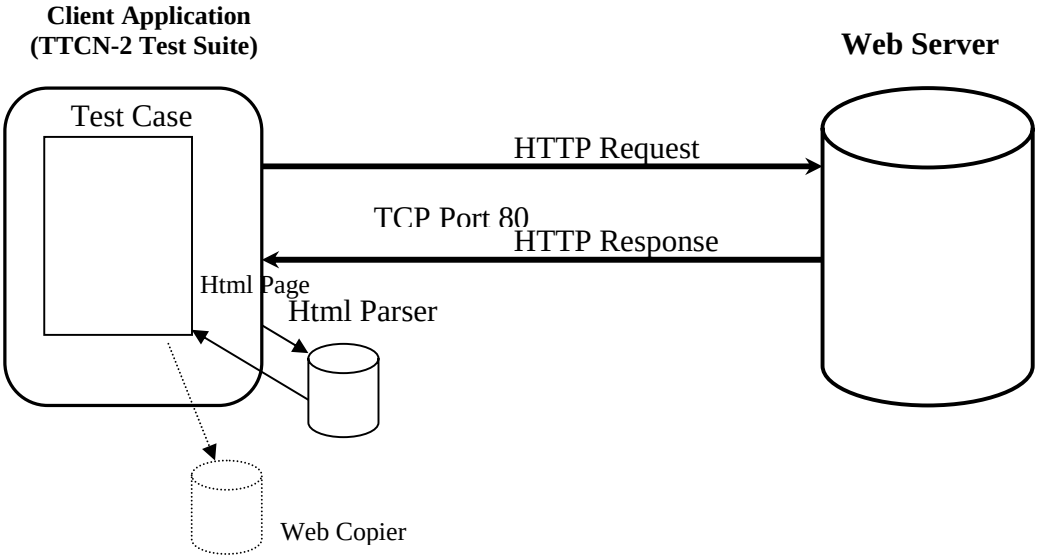


Figure 8: Test Scenario using TTCN-3

The figure above shows the test scenario when using TTCN-2 system. The client application sends request to the web server who sends the response back to the client. The client application is a TTCN-2 test suite which contains the test cases which contains the TTCN-2 code for sending and receiving messages. Unlike TTCN-3 test scenario, here Html parsing is not done through an external function(as this feature is not available in TTCN-2) but through an external application. The communication between the TTCN-2 test case and this external application is done via PIPE test port. This test port makes it possible to execute arbitrary UNIX commands directly from the test suite. During this thesis work, some effort was also made to make a Web Copier for TTCN-2 tool set and for this purpose a small application was made[1] which could do the task of fetching the resources from the web server and storing it on the hard disk thus making it possible for offline browsing.

Inorder to send the request for a resource, a client program uses the HTTP Test port which facilitates the message exchange. The test port establishes a TCP connection and sends to and receives the messages from the WebServer. The request/response message format is already described in the first section of this chapter.

6.3 Pure Test

Please see Chapter 4.3 which gives detailed explanation of the pure test tool set and its test scenario.

[1] Due to time restrictions only a very basic Web Copier was developed, but with whatever time spend on this work, it was concluded that a full fledge Web Copier can be made.

7 Tool Comparison

This chapter presents the advantages and disadvantages of the three technologies used in this thesis. It also compares the results of the tests done during the thesis work. A comparison and evaluation of all the three technologies with respect to web application testing is also presented.

7.1 Pure Test

This section presents the advantages and disadvantages of the Pure Test tool.

Advantages

- Graphical Tool

  Pure Test is written in Java and has a feel good graphical user interface(GUI). On the Solaris Operating Environment, the GUI has a Common Desktop Environment look and feel. On Microsoft Windows Environment, the GUI has a Microsoft Windows look and feel. All the commands are easily reached from the pull-downs available. See Appendix B for snapshots of the Pure Test tool.

- Easy to Install, Configure and Run

  Installing, configuring and running Pure Test is a trivial task. PureTest is distributed in a self-extracting format and installation is done by executing the installer and answering the questions about install location, etc. in the provided graphical user interface. On Unix installer is started by issuing command at the prompt: *sh puretest.bin.* Further details can be obtained from [11].

  Configuring PureTest is easy. Scenarios created using WebCrawler and HTTPRecorder can be loaded and run with a click of the button. To static check a website all that is needed is the website name, port number and client side proxy server name if any.

  PureTest is started using the generic launcher $INSTALL_HOME/bin/puretest. This is a script on UNIX and an executable program on Windows. Starting PureTest is accomplished by the following command: On UNIX: $INSTALL_HOME/bin/puretest & on Windows, use the PureTest shortcut.

- WebCrawler/HTTP Recorder

  These are the two excellent tools in Pure Test which assists in creating scenarios and static checking of the website. WebCrawler is a utility that crawls a web for static content and reports various statistics and error. The requests can be easily generated into PureTest scenarios for use in a function test session. The HTTP Recorder is a utility that eases the process of catching all requests between a browser and the web server.

- Excellent Feature List

  PureTest has a large number of features. Some of the features as per [11] are:

  - Verified on Solaris, Linux (RedHat) and Windows NT/2000/XP.
  - Capable of testing HTTP based applications including support for forms, uploading files, setting headers etc.
  - Possible to supervise applications based on standard protocols such as HTTP, NNTP, FTP, Telnet, SMTP, IMAP, JDBC (relational databases), LDAP, DNS and JMS (Java Messaging Service).
  - Create scenarios using the point and click interface. Includes a full-blown scenario debugger including single step, break points and response introspection.
  - Supports SSL v3
  - Standard session handling techniques, such as cookies and URL-rewriting is supported.
  - Support for parsing response codes or parsing page content for expected or unexpected strings.
  - API which enables developing scenarios for monitoring of arbitrary applications/protocols. Possible to create data driven Scenarios.
  - A simple command line interface is provided to execute scenarios, that can be used for automated testing.

- Reports

  The tool generated reports are shown in a user friendly format. Various metrics, broken links and the structure of the crawled web are shown in a format which is easy to understand and evaluate. Appendix shows a WebCrawler statistics window which shows the static check result.

- Prerequisites

  Not much knowledge of other fields is necessary as the tool is quite easy to install, configure and run. A person with a basic knowledge of operating systems can easily setup the tool and run tests. The lead time for testing thus becomes quite little.

- Free of Cost

  PureTest is completely free of cost. This is a great advantage for the companies/people who do not wish to invest in other commercial tools which are out there in the market for reasons like high cost, short testing needs etc.

- Debug Environment

  One of the great advantages of PureTest is that it has a debug execution environment which helps in response introspection and defects tracking.

Disadvantages

- Htmlparser

  Although PureTest is a well-evolved application, its Html parser still misses a few Html tokens. One would have expected the parser to be quite strong but PureTest misses out in this front. For example, an URL name containing '&' (ampersand in HTML) was missed during the parsing process.

- Load tests

  Load Tests cannot be performed in PureTest. Minq Software has another tool know as PureLoad. This tool is used for load testing but it is not free to use.

- Customisation

  It is not possible to customise PureTest according to the usage needs. For example, one of the usage requirements in this thesis work was to check for the possibility of having a web copier (off line browser) in the tools. PureTest does not have this feature.

7.2 TTCN-3

In this section, advantages and disadvantages of TTCN-3 is presented in context of Web Application testing.

Advantages

- Not very difficult to configure and run

  Configuring and running tests in TTCN-3 environment is not very difficult. However some knowledge of Unix is required. A person with no knowledge of Unix and TTCN will find it difficult to come to terms with various tools.

- Function testing.

  TTCN has been well proven standard for various types of testing. Function testing is one such area. Lot of projects in Ericsson have used TTCN for function testing. With its clean interface and structure it is good option for function testing a web application.

- Conformance testing

  Earlier versions of TTCN were basically designed for conformance testing. TTCN-3 keeps its strength from earlier versions intact. Tests can be easily done to check for the web client/server's conformance to various Internet protocols.

- Load Testing

  One of the new features in TTCN-3 when compared to earlier versions of TTCN is the support for load testing. Since load testing is always high on the testing requirements of any web application, TTCN-3 could be used for it. The language constructs and principles are quite powerful and even simple test cases written in TTCN-3 code for load testing could test the web application extensively.

- Programming Language

  TTCN-3 language is close to programming languages like C/C++. This makes it easier to design complicated test cases for complex systems. A person with programming background can easily utilize the language constructs in a powerful way thus using the might of the language. A person with programming background can easily get into the task of writing test scenarios in TTCN-3 thus reducing the lead time for the test phase when compared to the person with no programming background.

- Interface to other languages

  Although TTCN-3 is quiet powerful in its own respect, it lacks lot of features of a conventional programming language. There are times when it becomes extremely hard or even impossible to make test scenarios using TTCN-3 language. For this purposes, TTCN-3 has introduced the concept of external functions. Tool Vendors (in this case Ericsson Hungry) has incorporated this feature and introduced the support of C, C++ and Perl. With this support, it becomes possible to use TTCN-3 data types in the above languages. The external functions act as an interface between TTCN and the above programming languages. With this feature, complicated coding could be done using the other languages and results can be passed to TTCN-3 test suite, which could use the result for further processing.

Disadvantages

- Not suitable for a person with no knowledge of TTCN

  Using TTCN-3 for testing Internet Application requires that the tester or web master has some basic knowledge about TTCN notation. A person with no knowledge of the notation would find it difficult to test his web application.

- Flexibility

  The language is not flexible enough to perform various programming tasks, which are needed when performing automated website testing. Designing test cases for automated website testing, requires that the language has the possibility of creating complex code structure but during this thesis work, it was found that the language and the tool set is not flexible enough, yet. An example of inflexibility of tool set is the unsupported feature,

RegExp. The language has the support of RegExp, which could be of good help when making Html Parser, but Titan does not have the support for this feature yet. This support is planned for near future.

- WebCrawler

  WebCrawler cannot be made using only TTCN-3 language. TTCN-3 lacks flexibility of a pure programming language like C, C++. As automatic testing of a website requires the web pages to be parsed, TTCN-3 does not have enough language constructs which can facilitate this kind of string manipulation tasks. It does not have string function like strcmp, strncmp etc which are so essential when performing a matching/update/storing operation on plain text, which is the heart of a web crawler.

- Textual environment

  TTCN-3 environment is, at the moment text based. There are no graphical tools available but discussions with Rational/IBM [14] is going on to integrate the environment to Eclipse [14] thus giving it full fledge graphical interface. Presently, the tools are started by giving commands at the command prompt. The results are stored in a log file. It becomes really cumbersome when a big website is being tested as it generates lot of output.

- Html Parser.

  The HtmlParser made during this thesis work is not full fledge. Lot of functionality is still missing. It is just a simple application, which takes out urls from a html page. It can miss lot of tokens. The main reason for this ‘basic’ HtmlParser is the amounts of time spend on this work, which was not much, due to other priorities in this thesis work. The parser is not fast or robust as not much time was spent on the design and implementation. A big html file (ex:300kB) takes lot of time to parse. However small html files parses satisfactorily.

- Cost

  Licence cost for TTCN-3 tool set is at present free in Ericsson but looking at the other tool vendors, it costs quite a lot of money for yearly licences. If the project members do not have the knowledge of TTCN, it will costs the company some money and time to bring the workers to appropriate competence level.

7.3 TTCN-2

Advantages

- Good for function test

TTCN-2 has been well a proven standard for various types of testing. Function testing is one such area where over the years it has been used. Lot of projects in Ericsson have used or are using TTCN-2 for function testing

- Good for conformance test

  The Open Source Interconnection (OSI) model leaves the implementation of a certain protocol open. So there is always a question whether a certain implementation is correct and conforms to the relevant OSI protocol standard. In order to solve these questions, TTNC was introduced. Over the years, TTCN-2 has been well proven in this field. Many of the major companies like Ericsson, Nokia have used TTCN-2 for conformance testing of their applications.

- WebCopier

  Because of the better design of the HTTP adapter in the TTCN-2 tool set, it is trivial to implement a WebCopier compared to TTCN-3. There is no need to modify the adapter to have support for this feature.

Disadvantages

- Not suitable for a person with no knowledge of TTCN

  Using TTCN-2 for testing Internet Application requires that the tester or web master has some knowledge about the TTCN notation. A person with no knowledge of the notation would find it difficult to understand the concepts of the TTCN notation and will be unable to use the powers of the notation.

- Flexibility

  It was found during the thesis work that the language is not flexible enough to perform various programming tasks, which are needed when performing automated website testing. Designing test cases for automated website testing requires that the language has the possibility of creating complex code structure, but during this thesis work, it was found that the language and the tool set is not flexible enough. It does not have any string manipulation functions which are a prime requirement, if one needs to make a test case to perform automatic web site testing.

  TTCN-2 does not present any look and feel of a programming language thus making it difficult for any one trying to make complex test cases. Another feature missing from the language is the possibility of printing the values of any variables or log statements. Even to print a small log statement requires invocation of another test adapter taking up unnecessary execution time and resources.

- Load Testing

TTCN-2 is not designed for load testing which sometimes is the prime requirement of web application testing

- Htmlparser

  To make a WebCrawler in TTCN-2, the HtmlParser has to be implemented as an application, as the TTCN-2 notation does not provide enough language constructs to make a HtmlParser. To communicate with this outside application, a different test port (PIPE) has to be used.

- Cost

  The licence for the TTCN-2 tool set is free of cost in Ericsson but looking at the other tool vendors, it costs quiet a lot of money for yearly licences. Also, if the project members do not have the knowledge of TTCN, it will costs the company some money and time to bring the workers to an appropriate competence level.

7.4 Evaluation

Based on the criteria described in Chapter 1.1, Test Cases were written using TTCN-2 and TTCN-3. In the case of PureTest, 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 PureTest 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 PureTest. The main conclusions are presented below.

7.4.1 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 were 67, out of which 3 were erroneous.

	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 PureTest

From the above table, 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 '&'. The website which was tested had some pages containing ‘&’. Improving the TTCN HtmlParser to support this, enabled it to extract around 10 more resources. Looking closely at the above results it can be seen that the difference between TTCN-3 Html Parser and PureTest is actually the ‘ampersand’ problem. The column ‘text/html’ shows the variations. To look at the sample output from running the tests, check section 7.4.4 ‘User Interface’.

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

b. http://esekant027.epk.ericsson.se/019/

	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

c. http://esekant027.epk.ericsson.se/025/

	TTCN-3 Result	PureTest results
Total Pages	27	27
Total size	572 kB	572 kB
Cacheble Pages	27	27
Private Pages	0	0
Non-Cacheble Pages	0	0
Content Type		
Text/html	11	11
Images	7	7
Pdf	-	0
Multimedia	0	0
Others	0	0
Error	3	2
Total Time	0.7 s	2.0 s

d. http://esekant027.epk.ericsson.se/032/

	TTCN-3 Result	PureTest results
Total Pages	22	26
Total size	514 kB	534 kB
Cacheble Pages	22	26
Private Pages	0	0
Non-Cacheble Pages	0	0
Content Type		
Text/html	7	8
Images	15	18
Pdf	-	0
Multimedia	0	0
Others	0	0
Error	1	2
Total Time	0.6 s	4.0 s

e. http://esekant027.epk.ericsson.se/011/

	TTCN-3 Result	PureTest results
Total Pages	103	98
Total size	520 kB	505 kB
Cacheble Pages	103	98
Private Pages	0	0

Non-Cacheble Pages	0	0
Content Type		
Text/html	93	93
Images	10	5
Pdf	-	0
Multimedia	0	0
Others	0	0
Error	0	0
Total Time	11.0 s	6.0 s

f. http://esekant027.epk.ericsson.se/016/

	TTCN-3 Result	PureTest results
Total Pages	29	29
Total size	3228 kB	3228 kB
Cacheble Pages	29	29
Private Pages	0	0
Non-Cacheble Pages	0	0
Content Type		
Text/html	2	2
Images	7	7
Pdf	-	2
Multimedia	0	0
Others	20	18
Error	0	0
Total Time	0.3 s	4.0 s

g. http://esekant027.epk.ericsson.se/018/

	TTCN-3 Result	PureTest results
Total Pages	23	18
Total size	111 kB	96 kB
Cacheble Pages	23	23
Private Pages	0	0
Non-Cacheble Pages	0	0
Content Type		
Text/html	7	7
Images	15	10
Pdf	-	2
Multimedia	0	0
Others	1	1
Error	18	18
Total Time	1.0 s	1.5 s

It can be said from the above results that both TTCN and Pure Test give approximatly similar results and work is needed on improvement of their parsers.

7.4.2 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. PureTest 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. The difference in performance of the parsers can be seen when parsing html pages that are more than 200 kB. A page of size 50 kB parses in TTCN-3 as quickly as it doest in PureTest. A test was performed on a web page of size 330 kB. PureTest took 3-4 seconds to parse the contents and get the resources in it. HtmlParser in TTCN-3 took around 4 minutes. Result of section **b** in 7.4.1 is an example. This is a vast difference in performance but the factors are known and making the parser more optimized will solve the problem.

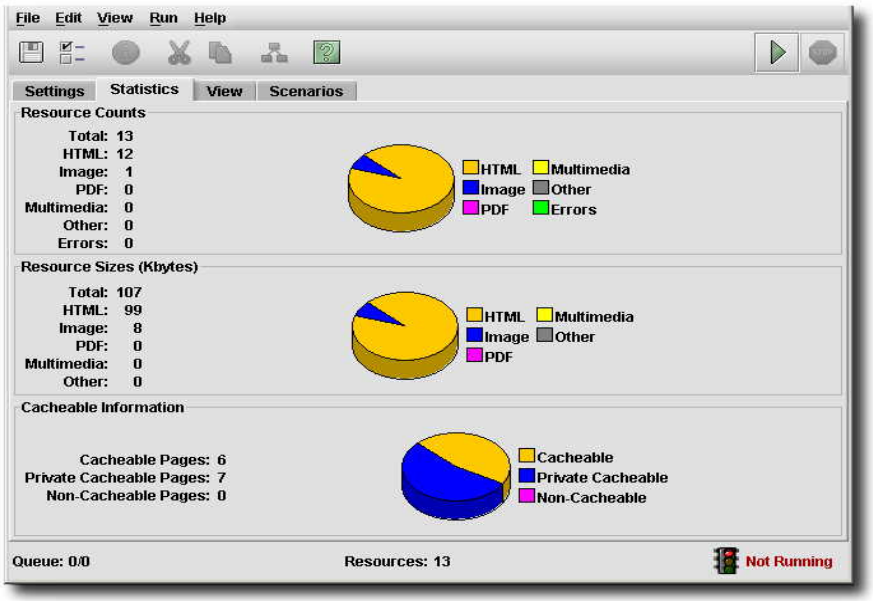
7.4.3 Speed

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

7.4.4 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. PureTest provides the results in graphs and structures, which are easy to understand and maintain. Below is the example of how Pure Test and TTCN present its results:

PureTest results



TTCN-3 Results (as printed in log file)

**********FINAL REPORT**********
Total Pages : 62
Total size : 3712542 bytes
cacheble Pages : 62
Private Pages : 0
Non-Cacheble Pages : 0
Content Type :
text/html : 40
media : 0
image : 13
other : 9
Error : 3
Total Time : 3.203638 s

Pure Test certainly scores over TTCN tools in this issue.

7.4.5 Debug Environment

PureTest has an excellent feature by which tests can be run in debug mode. When performing automated testing in PureTest, it is possible to set breakpoints, change the parameter values thus giving the tester a highly dynamic environment. TTCN-3 tests are static and cannot be changed during run time. A list of test cases has to be given before the execution starts and nothing can be

changed during runtime. In TTCN-2, values can be assigned dynamically during execution but it is not possible in the context of automated web tests, the reason being the complex nature of the test case.

7.4.6 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 licences. 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 PureTest (or for that matter many proprietary softwares). PureTest 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.

7.4.7 Function testing

TTCN-2 and TTCN-3 are well proven in this area and several projects in Ericsson have used them successfully. However, function testing a web application is a complex task and very much depends on the functionality of the application. In this thesis work, only a basic web site was tested and it was found that it is possible to function test a basic web site using TTCN-2 and TTCN-3. Complex web applications present greater challenges to the testers and at present due to time constraint and lack of good knowledge in this area, it can not be said how far TTCN-2/3 would succeed in function testing a complex web application. PureTest is promoted as a Function testing tool and performs on par with TTCN-2/3.

7.4.8 Load Testing

TTCN-3 contains some language enhancements compared to TTCN-2, which makes it highly powerful in the Load Testing area. Although not much work was done in this area during the thesis work, it can be said that TTCN-3 has a good potential in this area. PureTest on the other hand is not intended for load testing.

7.4.9 WebCopy

During the thesis work, some effort was made to check whether it is possible to make a web copier using the technologies. PureTest does not have a web-copier. The TTCN-3 tool set has some design issues that requires some changes in the HTTP protocol adapter. Solving these issues will make it possible to design test cases, which would enable web-copy. On the other hand, it is possible to make a web copier using the TTCN-2 tool set without any modification to the product. An effort was made to design a basic level test case, which would enable web copy and the output was a success. A full fledge web copier can be made using TTCN-2 tool set.

7.5 TTCN-3 vs TTCN-2

This section presents the comparison of the TTCN-3 and TTCN-2 languages and tool-sets with respect to Web Application testing.

It was observed during the thesis work that the TTCN-3 language is more flexible than TTCN-2. TTCN-3 is more of a programming language consisting of more language constructs than TTCN-2. In TTCN-2, certain language constructs have to be designed in their own section for example, types have to be declared in the declaration part, constraints in the constraint part, test cases in the dynamic part etc. This rule becomes quite restrictive when designing complicated test cases. TTCN-3 is far more flexible in this aspect. Variables, types, constants etc can also be declared in the test case declaration. Another aspect that is missing in TTCN-2 is the support for logging. Many times during designing test cases and performing test runs, it becomes essential to track variable values. This also helps in debugging errors. The only way to print a value of a variable or for that matter a message to track the progress or print the report of an automated web site test, a PIPE test port has to be used. This is a cumbersome process and just to print something involves a lot of work, eg.: writing some lines of TTCN-2 code, invoking a process for the test port etc. Invoking an extra test port for logging purposes also affects the execution time, processor power and the memory usage.

TTCN-3 has a language construct, ‘regexp’, which is similar to the language Perl. This is an excellent feature by which text manipulation work can be achieved. In the area of Automated Web Application testing, text manipulation techniques are very essential. With ‘regexp’ it is possible to make a WebCrawler in pure TTCN-3, which at the moment is not possible, as ‘regexp’ is not supported by TTCN-3 tool set, Titan. However, the support of ‘regexp’ is planned in near future deliveries of Titan. TTCN-2 language and the tool set do not have such a feature of ‘regexp’, apart from the fact that it does not have any important string manipulation functions. Therefore it can be said that it is not possible to make a WebCrawler using pure TTCN-2.

7.6 Final Comment

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.

Reference

Books

[1] Fewster, Mark and Graham, Dorothy; *Software Test Automation, Effective use of test execution tools;* 1999; Adison-Wesley, Harlow; ISBN 0-201-33140-3

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

Standards

[3] ISO/IEC 9646-3 (1998): "Information technology - Open systems interconnection – Conformance testing methodology and framework - Part 3: The Tree and Tabular Combined Notation (TTCN) Edition 2".

[4] ETSI ES 201 873-1 V2.2.0 (2002-03), Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language

Web Links

[5] Evaluating World Wide Web: A Global Study of Commercial Sites, 1997 by James Ho. http://www.ascusc.org/jcmc/vol3/issue1/ho.html

[6] Telelogic AB , www.telelogic.com

[7] Testing Technologies IST GmbH , www.testing-tech.de

[8] Danet GmbH , www.danet.de

[9] Minq Software AB, www.minq.se

[10] Scenario Editor User's Guide, http://www.minq.se/products/puretest/doc/index.html

[11] PureTest User's Guide, http://www.minq.se/products/puretest/doc/index.html

[12] Mercury Interactive, www.mercuryinteractive.com

[13] DoctorHtml, www.doctorhtml.com

[14] Rational/IBM , www.ibm.com

[15] RFC 2616 for HTTP , http://www.w3.org/Protocols/rfc2616/rfc2616.html

[16] LinkScan from Electronic Software Publishing Corporation www.elsop.com/linkscan/

[17] ParaSoftTechnologies www.parasoft.com

[18] IBM's Ration Robot www.ibm.com

[19] HT://Check http://htcheck.sourceforge.net/

[20] InSite http://insite.sourceforge.net/

[21] OpenSourceTesting http://opensourcetesting.org

Other Documents

[22] Thesis Proposal , Validation of Internet Application

Other Online Resources

http://www.softwareqatest.com
http://www.stickyminds.com
http://aptest.com/resources.html#web-source
http://www.w3.org/WAI/ER/existingtools.html
http://www.itworld.com/nl/ecom_in_act/03192003/
http://www.usablenet.com/accessibility_usability/webtesting.html

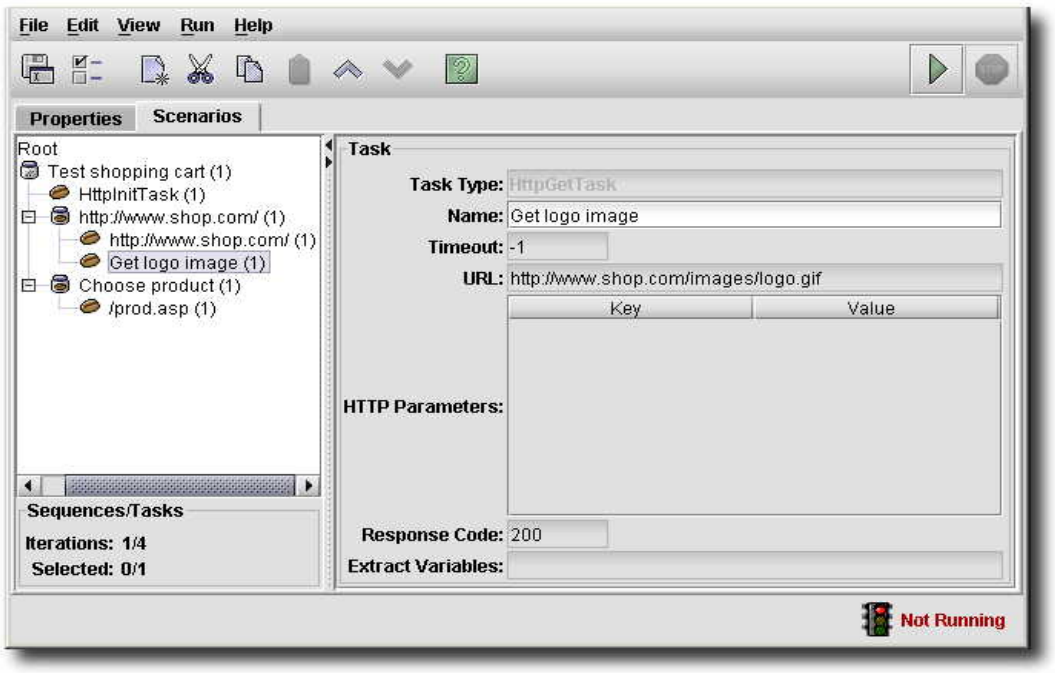
Glossary

API	Application Program Interface
ASN.1	Abstract Syntax Notation
ASP	Abstract Service Primitive
ATM	Asynchronous Transfer Mode
DECT	ETSI TDMA standard for Digital Enhanced Cordless Telephone
DHTML	Dynamic HTML
DNS	Domain Name Service
ExTeL	Executable Test Language
ETS	Executable Test Suite
FTP	File Transfer Protocol
GSM	Global System for Mobile communication
GUI	Graphical user interface
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IDL	Interface Definition Language
IE	Internet Explorer
INAP	Intelligent Network Application Protocol
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISUP	Integrated Services Digital Network User Part
IUT	Implementation Under Test
JDBC	Java Database Connectivity
JDK	Java Development Kit
JFC	Java Foundation Classes
JMS	Java Messaging Service
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
NNTP	Network News Transfer Protocol
OSI	Open System Interface
PCO	Point of control and observation
PDU	Protocol Date Unit
RDBMS	Relational Database Management System
SCS	System Certification System, Ericsson's tool set for TTCN-2
SMTP	Simple Mail Transfer Protocol

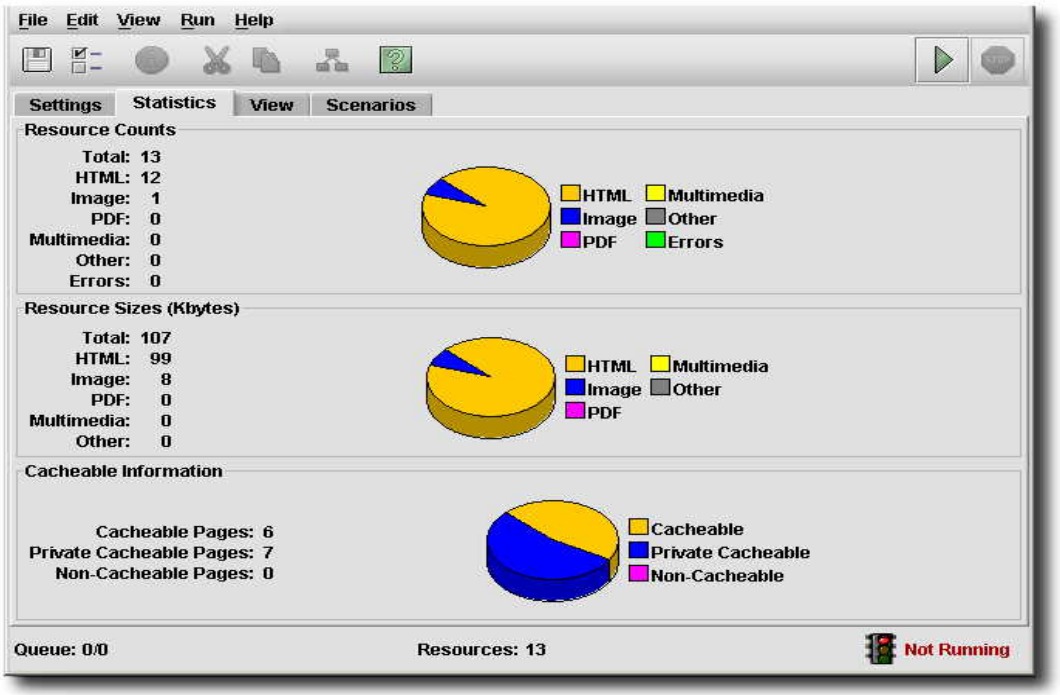
SSL	Secure Socket Layer
SS7	Signalling System No.7
TCE	Test Case Executor
TCP	Transmission Control Protocol
TTCN-2	Tree and Tabular Combined Notation
TTCN-3	Testing and Test Control Notation
URL	Uniform Resource Locator
UMTS	Universal Mobile Telecommunications System
3GPP	3rd Generation Partnership Project

Appendix A PureTest Snapshots

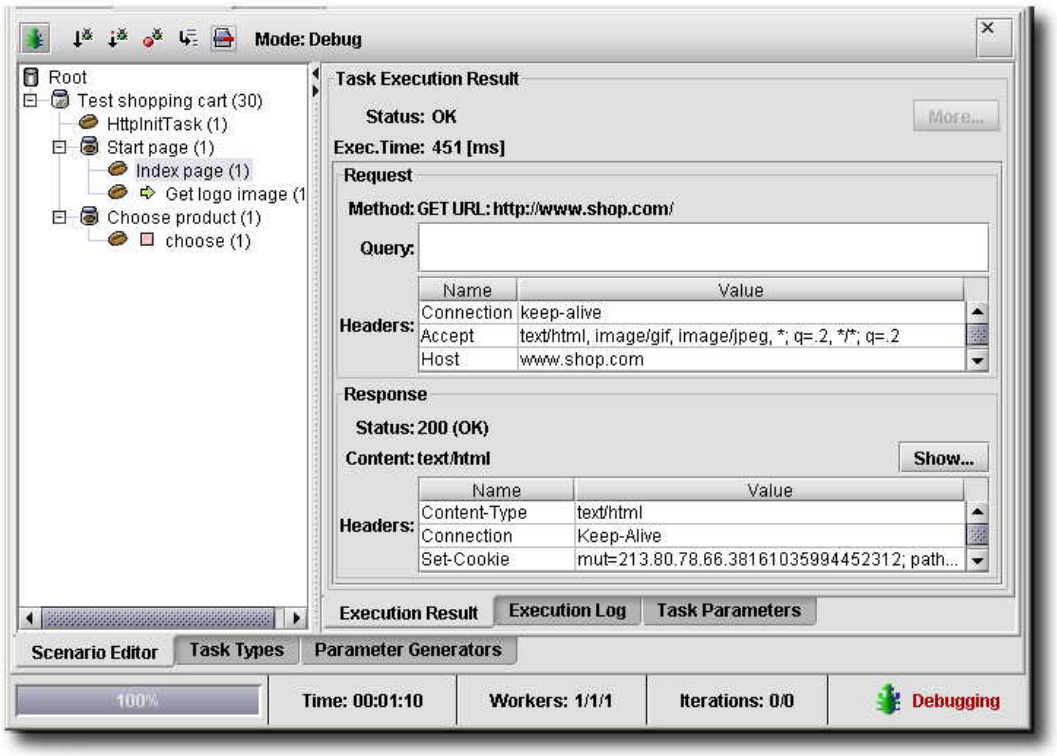
Scenario Editor



WebCrawler Result Window



Execution Window



Appendix B ITEX Editor

ASN.1 ASP Type Definition

ASP Name : EnterConference

PCO Type : PCO_Type

Comments :

Type Definition

```
SEQUENCE {
 conftype1 conftype,
 nametype1 nametype
}
```

Detailed Comments :

ASP Type Definition

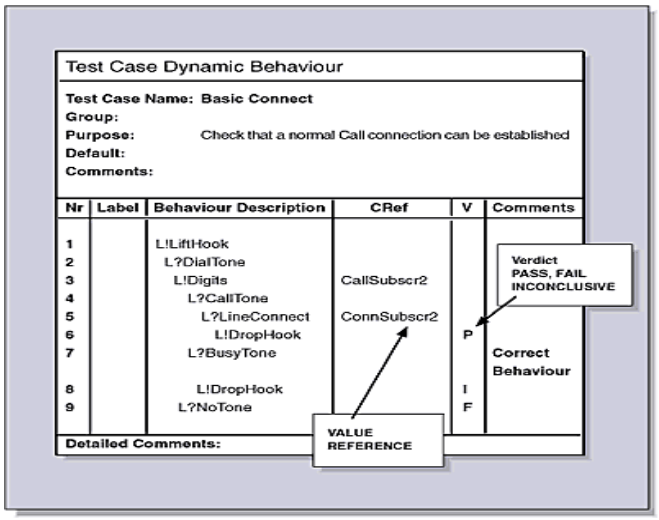
ASP Name : EnterConferenceT

PCO Type : PCO_Type

Comments :

Parameter Name	Parameter Type	Comments
conftype1	conftype	
nametype1	nametype	

Detailed Comments :



Appendix C TTCN General Terms

ASP, Abstract Service Primitive

An implementation-independent description of an interaction between a service-user and service-provider at an (N)-service boundary. [1]

Abstract test case

A complete and independent specification of the actions required to achieve a specific test purpose, defined at the level of abstraction of a particular abstract test method. It includes a preamble and a postamble to ensure starting and ending in a stable state (i.e., a state which can be maintained almost indefinitely, such as the "idle" state or "data transfer" state) and involves one or more consecutive or concurrent connections.

CP, Coordination Point

Communication between test components in the lower tester is achieved via coordination points (CP). Upper Tester test components may communicate with each other via CPs[2]

PDU, Protocol Data Unit

Information that is delivered as a unit among peer entities of a network and that may contain control information, address information, or data. In layered systems, a unit of data that is specified in a protocol of a given layer and that consists of protocol control information of the given layer and possibly user data of that layer.

PCO, Point of control and observation

A point within a testing environment where the occurrence of test events is to be controlled and observed, as defined in Abstract Test Method.[1]

MTC, Master test component

There must be at least one test component always present in the test system. This is called the master test component and it is responsible for coordinating and controlling the test and for setting the final verdict of the test.[2]

PTC, Parallel test component

[1] CCITT X.290. 1992. OSI conformance testing methodology and framework for protocol recommendation for CCITT applications –General concepts. International Telecommunications Union (ITU-T)
[2] Wiles, A.1993 The Tree and Tabular Combined Notation – A Tutorial.Uppsala. Telia Research F93 1053 Version 1.2 (SC 3)

Test components affecting to certain service access point and controlled by MTC.

LT, Lower Tester

The representation in Recommendations X.290 to X.294 of the means of providing, during test execution, indirect control and observation of the lower service boundary of the IUT via underlying service provider.

UT, Upper Tester

The representation in Recommendations X.290 to X.294 of the means of providing, during test execution, control and observation of the upper service boundary of the IUT, as defined by the chosen Abstract Test Method

Appendix D OSEK

Increased costs in design, integration and production test process of embedded software for automative systems have become a major problem over last couple of decades. This has led to a European automotive industry standards effort to produce open systems interfaces for vehicle electronics. The full name of the project is OSEK/VDX. OSEK is an acronym formed from a phrase in German, which translates as "Open Systems and Corresponding Interfaces for Automotive Electronics". VDX is based on a French standard (Vehicle Distributed eXecutive) which has now been merged with OSEK. The OSEK group is managed by a steering committee representing Adam Opel, BMW, DaimlerChrysler, University of Karlsruhe, PSA, Renault, Robert Bosch, Siemens, and Volkswagen.

The open architecture introduced by OSEK/VDX comprises the three areas:

- Communication (Data exchange within and between Control Units), OSEK COM.
- Network Management (Configuration determination and monitoring), OSEK NM and
- Operating System (Real-time executive for ECU software and basis for the other OSEK/VDX modules). OSEK OS.

OSEK NM defines protocols for managing in-vehicle networks. OSEK COM defines a protocol for inter-ECU communications on in-vehicle networks, typically Controller Area Network (CAN). In automotive applications no ECU is an island. The OSEK COM standard recognizes this and attempts to ensure that all ECUs in a system can interwork efficiently. A number of companies have produced libraries of driver code that implement communications through a mechanism conformant with the OSEK COM standard. OSEK OS defines a standard operating system interface for in-vehicle ECU applications. Alongside the OSEK OS specification there is also a standardized OSEK Implementation Language (OIL) that will let the developer specify the details of the application area in a special configuration file. An OIL tool is then able to read a file in OIL format that describes the application structure and configure the OS so that it is scaled to the demands of the application. This allows a better use of ROM vs. RAM due to the ability to calculate data off-line rather than at run-time. This is important because RAM is much more expensive than ROM in microcontrollers typically used in the automotive industry.

More information about OSEK/VX can be obtained from :

http://www.osek-vdx.org/ and

http://www.sovereign-publications.com/auto-art-3.htm

Appendix E Behaviour Trees

In TTCN-2, dynamic behaviour table contains the exeuction logic of the test case. Give below is the sample dynamic behaviour table.

Default Dynamic Behaviour					
Default Name : *DefaultId&ParList*					
Group : *DefaultGroupReference*					
Objective : *FreeText*					
Comments : *[FreeText]*					
Nr	**Label**	**Behaviour Description**	**Constraint Ref**	**Verdict**	**Comments**
1	.	.	.	.	.
2	.	.	.	.	.
.	.	*StatementLine*	.	.	.
.	*[Label]*	.	*[ConstraintReference]*	*[Verdict]*	*[FreeText]*
.	.	.	.	.	.
		TreeHeader			.
.	.	*StatementLine*	.	.	.
n	.	.	.	.	.
Detailed Comments: *[FreeText]*					

The behaviour description column of a dynamic behaviour table contains the combination of TTCN statements that are to be executed for the test case. The set of these combinations is called the Behaviour Tree. Each TTCN statement is a node in the behaviour tree.
Each TTCN statement in the behaviour description has to be on a separate line. The statements can be related to one another in two ways :

- as sequences of TTCN statements
- as alternative TTCN statements.

Sequences of TTCN statements are written one statement line after the other, each new TTCN statement is indented once from left to right, with respect to its predecessor.

```
EVENT_A
   CONSTRUCT_B
   EVENT_C
```

Statements at the same level of indentation and belonging to the same predecessor node correspond to the possible alternative statements which may occur at that time.

```
CONSTRUCT_A1
STATEMENT_A2
EVENT_A3
```

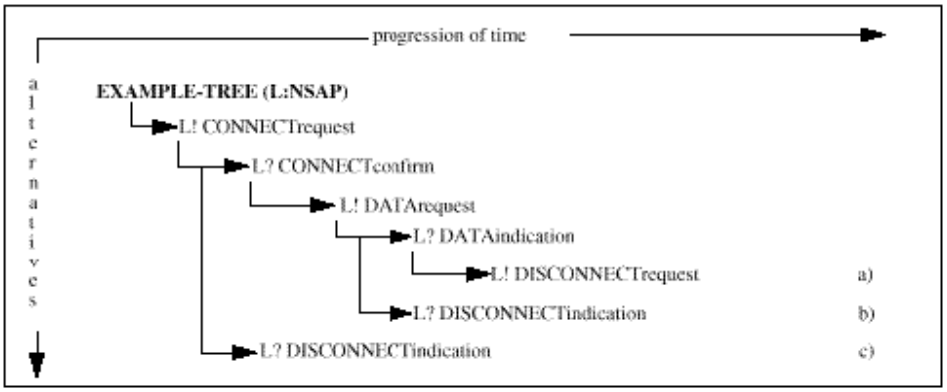
Execution of Behaviour Tree :

Lets say that the following sequence of events can occur during a test whose purpose is to establish a connection, send and receive some data and close the connection. The event occurs at lower tester :

a) CONNECTrequest, CONNECTconfirm, DATArequest, DATAindication, DISCONNECTrequest;

The IUT or the service-provider can stop progress at any time. This generates two more sequences:
b) CONNECTrequest, CONNECTconfirm, DATArequest, DISCONNECTindication;
c) CONNECTrequest, DISCONNECTindication.

The three sequences of events can be expressed as a TTCN behaviour tree. There are five levels of alternatives, and only three leaves (a to c), because the SEND events L! are always successful. Execution is to progress from left to right (sequence), and from top to bottom (alternatives). The following figure illustrates this progression, and the principle of the TTCN behaviour tree:



There are no lines, arrows or leaf names in TTCN. The behaviour tree of the previous example would be represented as follows:

Test Step Dynamic Behaviour					
Test Step Name	:	TREE_EX_1 (L:NSAP)			
Group	:	TTCN_EXAMPLES/TREE_EXAMPLE_1/			
Objective	:	To illustrate the use of trees.			
Defaults	:				
Comments	:	Note - This example can be simplified by using Defaults.			
Nr	**Label**	**Behaviour Description**	**Constraint Ref**	**Verdict**	**Comments**
1		L ! CONNECTrequest	CR1		Request…
2		L ? CONNECTconfirm	CC1		… Confirm
3		L ! DATArequest	DTR1		Send Data
4		L ? DATAindication	DTI1		Receive Data
5		L ! DISCONNECTrequest	DSCR1	PASS	Accept
6		L ? DISCONNECTindication	DSCI1	INCONC	Premature
7		L ? DISCONNECTindication	DSCR1	INCONC	Premature
Detailed Comments:					

More details on execution behaviour, constraints, rules and details about other concepts of TTCN can be found from the TTCN-2 standard [3]

Appendix F TTCN-3 Test Suite

```
/
//HTTP_Test.ttcn
//
//Main TTCN-3 module for HTTP web testing.
//
//Nisar.Lalani@ericsson.com
//

module HTTPTest {


//Import ASPs
import from HTTPASP all;


//External Function implemented in C++. This funciton parses the html code
//and returns urlList and number of outgoing links
//Parameters :
//    body    : htmlbody to be parsed
//    hostName: hostName
//    urlList : list of resrouces in the htmlbody
//    directory         : used to generate a proper path name for the urls
//    startingDirectory : used to generate a proper path name for the urls
//              example :../abc.html will get changed to http://ab.com/abc.html

external function htmlParser(charstring body,
                                                        charstring hostName,
                                                        inout headChar urlList,
                                                        charstring directory,
                                                        charstring startingDirectory,
                                                        inout integer outLinks);


// Types used in test suite

type record of charstring headChar;

type record outPut{
        charstring mainPage,
        headChar   links,
        integer outGoingLinks
}

type record of outPut finalOutPut;

//structure used to store content type
type record Content{
        integer textHtml,
        integer media,
        integer image,
        integer other
}
```

```
//struture used to keep track of the website statistics
type record stats{
	integer totalPages,
	integer totalSize,
	integer cacheCnt,
	integer privateCacheCnt,
	integer nonCacheCnt,
	Content contentType,
	integer errorPages
}

type component client {
	port HTTPPort http;
	var headChar urlProcessed := {};
	var finalOutPut completeList := {};
	var stats ts := {
	totalSize := 0,
	cacheCnt := 0,
	privateCacheCnt := 0,
	nonCacheCnt :=0
	}
};

//ASP templates
template HalfClose h := {};
template Message recv_resp := { response := * };

template Message rec_ttcn_site(template Response rec) := {
	response := rec
}

template Message get_ttcn_site( template Request req ) := {
	request := req
}

//request url
template Request requestUrl (HeaderLines headerFields) := {
	method := "GET",
	//uri    := "http://esekant027.epk.ericsson.se/019/",
		uri := "http://esekant027.epk.ericsson.se/042/",
	//uri :=
"http://esekant027.epk.ericsson.se/042/General%20TCM%20Information/General%20TCM%20Information.shtml",

	major  := 1,
	minor  := 1,
	header := headerFields,
	body   := omit
}

template Request requestEsekant (HeaderLines headerFields , charstring url) := {
	method := "GET",
	uri    := url,
	major  := 1,
	minor  := 1,
	header := headerFields,
```

```
        body   := omit
}

//http response
template Response rec := {
        major := 1,
        minor := 1,
        statusCode := 200,
        statusText := "OK",
        header := ?,
        body  := *
}

//In some cases, 'Ok' is recieved instead of 'OK'
template Response recWithSmallOk := {
        major := 1,
        minor := 1,
        statusCode := 200,
        statusText := "Ok",
        header := ?,
        body  := *
}


//bad request : 404 : Object Not Found
template Response badRec := {
        major := 1,
        minor := 1,
        statusCode := *,
        statusText := *,
        header := ?,
        body  := *
}

//Template for Connet
template Connect conn(charstring host, integer portNo) := {
        hostName := host,
        portNumber := portNo
}

//Prints the final report
function finalReport(inout stats finalStats,float totalTime){

        log("\n***************FINAL REPORT********************* \n",
                "Total Pages       : ",finalStats.totalPages , "\n",
                "Total size        : ",finalStats.totalSize , " bytes\n" ,
                "cacheble Pages    : ",finalStats.cacheCnt, "\n",
                "Private Pages     : ",finalStats.privateCacheCnt , "\n",
                "Non-Cacheble Pages : ",finalStats.nonCacheCnt, "\n",
                "Content Type : " , "\n",
                "           text/html : " , finalStats.contentType.textHtml , "\n",
                "           media     : " , finalStats.contentType.media , "\n",
                "           image     : " , finalStats.contentType.image , "\n",
                "           other     : " , finalStats.contentType.other , "\n",
                "Error             : ",finalStats.errorPages , "\n",
                "", "\n",
```

```
                "", "\n",
                "", "\n",
                "Total Time        : ",totalTime, " s\n",
                "\n***********************************");
}

//Test Case Stats
function testCaseReport(integer bodylen ,
                                                charstring contentType ,
                                                charstring cache,
                                                float tcTimer,
                                                charstring url){
        log("\n***************Test Case Report*************** \n",
                "Url          : ",url, "\n",
                "Body Size    : ",bodylen , " bytes\n" ,
                "Content Type  : ",contentType, "\n",
                "Cacheble     : ",cache, "\n",
                "Response Time : ",tcTimer, " s",
                "\n***********************************");
}


//Function for error report
function testCaseFailReport(charstring url ,
                                                        integer statusCode,
                                                        charstring statusText ,
                                                        float tcTimer){
        log("\n***************Error Report*************** \n",
                "Url          : ",url, "\n",
                "StatusCode    : ",statusCode , " \n" ,
                "StatusText    : ",statusText, "\n",
                "Response Time : ",tcTimer, " s",
                "\n***********************************");
}


function totalScenario(finalOutPut completeList){
        var integer rlen := sizeof(completeList);
        log("-----------------------------------TotalScenario------------------------------ \n");
        for(var integer i:= 0; i < rlen;i:=i+1){
                var integer len := sizeof(completeList[i].links);
           log(completeList[i].mainPage , "   Outgoing Links : ",completeList[i].outGoingLinks);
           for(var integer j:= 0; j < len; j:= j+1){

                        log("  ----   ", completeList[i].links[j]);
                }


        }
   log("-------------------------------------------------------------------------------\n");
}


function headerParsing(headChar responseHeader,
                                        inout charstring contentType,
                                        inout charstring contentLength) return charstring {
```

```
	//size of header
	var integer len := sizeof(responseHeader);
	//by default every page is cacheble
	var charstring cache := "cacheble"

	//go through the header
	for(var integer i:= 0;i < len ; i := i+1){
		var charstring arr;
		var charstring ch := substr(responseHeader[i],0,14);

		//length of this header
		var integer chlen := lengthof(responseHeader[i]);
		//If content length is given, store it in 'contentLength'
		if(substr(responseHeader[i],0,14) == "Content-Length"){
			var integer k := 0;
			//get the value
			for(var integer t:= 16; t < chlen; t:= t+1){
				arr[k] := responseHeader[i][t];
			   k := k+1;
			}
			contentLength := arr;
			log("Content-Length   ",contentLength);
		}else if(substr(responseHeader[i],0,12) == "Content-Type"){
			//If content Type is given, store it in 'contentType'
			var integer k:= 0;
			//get the value
			for(var integer t:= 14; t< chlen; t:= t+1){
				arr[k] := responseHeader[i][t];
					k := k+1;
			}
			log("Content-Type  ",arr);
			contentType := arr;
		}else if(substr(responseHeader[i],0,13) == "Cache-control"){
			//If Cache-Control is give,store the 'cache'
			var integer k:= 0;
			//get the value
			for(var integer t:= 15; t< chlen; t:= t+1){
				arr[k] := responseHeader[i][t];
				k := k+1;
			}
			log("Cache-control  ",arr);
			cache := arr;
		}

	}

	return cache;

}

function repeatFunction(charstring host,
                                                integer portNo,
                                                HeaderLines headerFields,
                                                charstring url,
                                                inout stats finalStats,
```

```
                                                    inout float totalTime,
                                                    charstring directory,
                charstring startingDirectory) runs on client {

timer t_guard := 10.0;

map(mtc:http, system:http);

var float responseTime;
var integer success := 0;
var integer err404 := 0;
var Message t3rec;
var Close c := {};
var charstring realUrl := "http://esekant027.epk.ericsson.se/042/"+url;
//Send Connect ASP
http.send(conn(host,portNo));

//add the url to the list
var integer elements := sizeof(urlProcessed);
urlProcessed[elements] := url;

var integer comListElem := sizeof(completeList);
completeList[comListElem].mainPage := url;
completeList[comListElem].links :={};
completeList[comListElem].outGoingLinks := 0;

//Send the request for a page
http.send(get_ttcn_site(requestEsekant(headerFields,url)));
t_guard.start;
alt{
[] http.receive(Message:rec_ttcn_site(rec))-> value t3rec{
        http.send(Close:c);
        responseTime := t_guard.read;
        success := 1;
        t_guard.stop;
        setverdict(pass);
}
[] http.receive(Message:rec_ttcn_site(recWithSmallOk))-> value t3rec{
        http.send(Close:c);
        responseTime := t_guard.read;
        success := 1;
        t_guard.stop;
        setverdict(pass);
}
[] t_guard.timeout{
        http.send(Close:c);
        setverdict(fail);
}
[] http.receive (Message:rec_ttcn_site(badRec))-> value t3rec{
        err404 := 1;
        http.send(Close:c);
        responseTime := t_guard.read;
        t_guard.stop;
        setverdict(fail);
}
[] http.receive {
```

```
		http.send(Close:c);
		responseTime := t_guard.read;
		t_guard.stop;
		setverdict(fail);
}
};
unmap(mtc:http, system:http);

//If we received what we expected
if(success){
		//update the totalPages
		finalStats.totalPages := finalStats.totalPages + 1;
		var charstring contentType := "";
		var charstring contentLength := "";
		//store the header in a variable
		var headChar responseHeader := t3rec.response.header;
		//Parse the header and get the contentLength,contentType and cache
		var charstring cache := headerParsing(responseHeader,
                                                                                contentType,
                                                                                contentLength);
		var charstring body := "";
		var integer bodylen := 0;

		//checking of the body is not done if the Content-Type is other
		//than text/html.If the body is html then length will be the
		//size of the body otherwise it will be the number specified
		//in 'Content-Length' field in the header
		if(ispresent(t3rec.response.body)){
			body := t3rec.response.body;
		}
		if(lengthof(body) > 1){
			bodylen := lengthof(body);
		}else{
			bodylen := str2int(contentLength);
		}
		//Print the report for this testcase
		testCaseReport(bodylen,contentType,cache,responseTime,url);

		//If html body is present, parsing of the body is necessary
		//to find the urls in them
		if(ispresent(t3rec.response.body)){
			body := t3rec.response.body;

			var headChar urlList := {};
			//var charstring startingDirectory := "http://esekant027.epk.ericsson.se";
			var integer outLinks := 0;
			//parse the html code to fetch the urls in them
			//body = html code , host = the host(ex: esekant027.epk.ericsson.se)
			//urlList = the resultant list of urls.
			//directory = the current directory level
			//startingDirectory = the starting level

			htmlParser(body,host,urlList,directory,startingDirectory,outLinks);
			completeList[comListElem].outGoingLinks := outLinks;
			//count the number of urls
			var integer len:= sizeof(urlList);
```

```
        if(len){
                //just print the list
                for(var integer i:= 0; i < len;i:=i+1){
                        log(urlList[i]);
                        var integer elem := sizeof(completeList);
                   completeList[elem-1].links[i] := urlList[i];
                }
                for(var integer i:= 0; i < len;i:=i+1){
                        //Check if this link is already processed. If yes, skip it, else
                        //process it
                        var integer lenUrlProcessed := sizeof(urlProcessed);
                        var integer alreadyProcessed := 0;
                        for(var integer k:= 0; k < lenUrlProcessed ; k:= k+1){
                                if(urlList[i] == urlProcessed[k]){
                                        alreadyProcessed := 1;
                                }
                        }
                        if(alreadyProcessed == 0){
                        //The code below fetches the current directory level
                        //from the url.
                        //ex: url = http://esekant027.epk.ericsson.se/042/sample/samp.html will
                        //result in http://esekant027.epk.ericsson.se/042/sample/
                        //This is passed to the 'repeatFunction'
                        var charstring tmpDir := "";
                        tmpDir := urlList[i];
                        var integer urlLen := lengthof(urlList[i]);
                   var integer exitLoop := 0;

                   for(var integer j:= urlLen; j != 0;j := j-1){
                                if(exitLoop == 0){
                                        if(tmpDir[j-2] == "/"){
                                                tmpDir[j-1] := {};
                                                exitLoop := 1;
                                        }
                                }
                                if(exitLoop == 0){
                                        tmpDir[j-1] := {};
                                }
                        }
                        //run the test for this url
                        repeatFunction(host,
                                                portNo,
                                                headerFields,
                                                urlList[i],
                                                finalStats,
                                                totalTime,
                                                tmpDir,
                                                startingDirectory);
                        }
                }
        }
}

//Update the counters
if(cache == "private"){
        finalStats.privateCacheCnt := finalStats.privateCacheCnt +1;
```

```
		}else if(cache == "no-cache"){
			finalStats.nonCacheCnt := finalStats.nonCacheCnt +1;
		}else if(cache == "cacheble"){
			finalStats.cacheCnt := finalStats.cacheCnt +1;
		}
		if(contentType == "text/html"){
			finalStats.contentType.textHtml := finalStats.contentType.textHtml + 1;
		}else if(substr(contentType,0,5) == "image"){
			finalStats.contentType.image := finalStats.contentType.image +1;
		}else {
			finalStats.contentType.other := finalStats.contentType.other +1;
		}

		finalStats.totalSize := finalStats.totalSize + bodylen;
		//testCaseReport(bodylen,contentType,cache,responseTime);
	}else{
		//If error occured
		if(err404 == 1){
			var charstring statusText := t3rec.response.statusText;
			var integer statusCode := t3rec.response.statusCode;
			testCaseFailReport(url,statusCode,statusText,responseTime);
			finalStats.errorPages := finalStats.errorPages +1 ;
		}else{
			var charstring statusText := "Error other than 'Object Not Found'";
			var integer statusCode := null;
			testCaseFailReport(url,statusCode,statusText,responseTime);
			finalStats.errorPages := finalStats.errorPages +1 ;
		}
	}
}

}

module HTTPTestCases{

	//Module Parameters
modulepar {
	charstring hostName;
	charstring startingUrl;
	charstring startingDirectory;
	headChar url
}

	//import from HTTPASP all;
import from HTTPTest all;

testcase simple() runs on client
{
	map(mtc:http, system:http);
	//var charstring host := "esekant027.epk.ericsson.se";
	//var charstring host := "ttcn.ericsson.se";
	var charstring host := "www.r.eth.ericsson.se";
	var integer portNo := 80;
	var HeaderLines headerFields := {"Host: epkws1238", "Connection: close"};
	http.send(conn(host,portNo));
	http.send(get_ttcn_site(requestUrl(headerFields)));
```

```
		alt{
		[] http.receive(Message:rec_ttcn_site(rec)){
			setverdict(pass);
		}
		[] http.receive{
			setverdict(fail);
		}
		};
		unmap(mtc:http, system:http);
}

// Test case : tc_static_check
// Purpose   : This test case crawls through the website and generates report
//		report on static content of the website
// Parameters:
//		finalStats : maintains various statistics of website
//		totalTime  : maintains total execution time
//
testcase tc_static_check(inout stats finalStats,inout float totalTime,charstring directory) runs on client
{

		//Initial timer
		timer executionTimer:=60.0
		//start the timer
		executionTimer.start;

		//Map main test component to system component
		map(mtc:http, system:http);

		//Timer for the test case
		timer t_guard := 10.0;

		var float responseTime;
		var integer success := 0;
		var integer err404 := 0;
		var charstring host := hostName;

		var integer portNo := 80;
			var charstring firstUrl := startingUrl;
		var HeaderLines headerFields := {"Host: 136.224.74.26", "Connection: close"};
		var Close c := {};
		var Message t3rec;

		//Send connect request to the webserver
		http.send(conn(host,portNo));

		//Send request to get a webpage
		http.send(get_ttcn_site(requestUrl(headerFields)));

		//Put the startingUrl to a list
		var integer elements := sizeof(urlProcessed);
		urlProcessed[elements] := firstUrl;

		//Maintain a list containing all the resources fetched.
		var integer finEle := sizeof(completeList);
		completeList[finEle].mainPage := firstUrl;
```

```
completeList[finEle].links := {};
completeList[finEle].outGoingLinks := 0;

t_guard.start;
alt{
[] http.receive(Message:rec_ttcn_site(rec)) -> value t3rec{
        http.send(Close:c);
        responseTime := t_guard.read;
        t_guard.stop;
        setverdict(pass);
        success := 1;
}

[] t_guard.timeout{
        http.send(Close:c);
        setverdict(fail);
}
[] http.receive(Message:rec_ttcn_site(badRec)) -> value t3rec{
        err404 := 1;
        http.send(Close:c);
        responseTime := t_guard.read;
        t_guard.stop;
        setverdict(fail);
}
[] http.receive {
        http.send(Close:c);
        responseTime := t_guard.read;
        t_guard.stop;
        setverdict(fail);
}
};
unmap(mtc:http, system:http);

//If the test passed
if(success){
        //Increment the totalPages counter
        finalStats.totalPages := finalStats.totalPages + 1;
        var charstring contentType := "";
        var charstring contentLength := "";
        var headChar responseHeader := t3rec.response.header;
        //Parse the http header from the response
        var charstring cache := headerParsing(responseHeader,
                                                                        contentType,
                                                                        contentLength);

        var charstring body := "";

        var integer bodylen := 0;

        //If body is present in the response, assign it to variable 'body'.
        if(ispresent(t3rec.response.body)){
                body := t3rec.response.body;
        }

        //If body is not present,take the length from 'contentLength' header
        //field. This is done for resources which are not of type text/html
```

```
//The body is not fetched when the resource is other than text/html
if(lengthof(body) > 1){
	bodylen := lengthof(body);
}else{
	bodylen := str2int(contentLength);
}

//Generate testcase report
testCaseReport(bodylen,contentType,cache,responseTime,startingUrl);

//If body is present, parse it and crawl
if(ispresent(t3rec.response.body)){
	body := t3rec.response.body;

	var headChar urlList := {};

	//variable to count outgoing links in this page.
	//Similar variables can be added if needed to keep track of other
	//things like inline resource in this page etc.
	var integer outLinks := 0;

	//html parser. This will return a list of the resources and
	//number of outgoing links in this page.
	htmlParser(body,host,urlList,directory,startingDirectory,outLinks);

	//increment the outgoinglink counter
	completeList[finEle].outGoingLinks :=outLinks;
	var integer len := sizeof(urlList);

	//if there are any resources in the list,fetch it from webserver
	if(len){
		//populate completeList with all the urls fetched from the
		//page
		for(var integer i:= 0; i < len;i:=i+1){
			log(urlList[i]);
		   var integer finEle := sizeof(completeList);
		   completeList[finEle-1].links[i] := urlList[i];
		}

		for(var integer i := 0; i < len;i := i + 1) {
			var integer lenUrlProcessed := sizeof(urlProcessed);
			var integer alreadyProcessed := 0;
			//Check if we have already processed the url
			for(var integer k := 0; k < lenUrlProcessed ; k := k+1){
				if(urlList[i] == urlProcessed[k]){
					alreadyProcessed := 1;
				}
			}

			//If the url has not been processed, request it from the
			//webserver
			if(alreadyProcessed == 0){
			var charstring tmpDir := "";
			tmpDir := urlList[i];
			var integer urlLen := lengthof(urlList[i]);
		   var integer exitLoop := 0;
```

```
                                        //get the directory of the url
                                    for(var integer j:= urlLen; j != 0;j := j-1){
                                            if(exitLoop == 0){
                                                    if(tmpDir[j-2] == "/"){
                                                            tmpDir[j-1] := {};
                                                            exitLoop := 1;
                                                    }
                                            }
                                            if(exitLoop == 0){
                                                    tmpDir[j-1] := {};
                                            }
                                    }

                                    //request the url from the server
                                    repeatFunction(host,
                                                                    portNo,
                                                                    headerFields,
                                                                    urlList[i],
                                                                    finalStats,
                                                                    totalTime,
                                                                    tmpDir,
                                                                    startingDirectory);
                                    }
                            }
                    }
            }

            //Update cache statistics
            if(cache == "private"){
                    finalStats.privateCacheCnt := finalStats.privateCacheCnt +1;
            }else if(cache == "no-cache"){
                    finalStats.nonCacheCnt := finalStats.nonCacheCnt +1;
            }else if(cache == "cacheble"){
                    finalStats.cacheCnt := finalStats.cacheCnt +1;
            }

            //update content type statistics
            if(contentType == "text/html"){
                    finalStats.contentType.textHtml := finalStats.contentType.textHtml + 1;
            }else if(substr(contentType,0,5) == "image"){
                    finalStats.contentType.image := finalStats.contentType.image +1;
            }else {
                    finalStats.contentType.other := finalStats.contentType.other +1;
            }

            //Update totalSize of the whole website
            finalStats.totalSize := finalStats.totalSize + bodylen;

    }else{
            //If 404 error received,produce a report
            if(err404 == 1){
                    var charstring statusText := t3rec.response.statusText;
                    var integer statusCode := t3rec.response.statusCode;
                    testCaseFailReport(startingUrl,statusCode,statusText,responseTime);
                    finalStats.errorPages := finalStats.errorPages + 1;
```

```
		}else{
			//If error other than 404
			var charstring statusText := "Error other than 'Object Not Found'";
			var integer statusCode := null;
			testCaseFailReport(startingUrl,statusCode,statusText,responseTime);
			finalStats.errorPages := finalStats.errorPages +1 ;
		}

	}

	//update totalExecution time
	totalTime := totalTime + executionTimer.read;
	executionTimer.stop;

	//print the whole scenario
	totalScenario(completeList);

}

/* Test case : tc_basedon_module_params  */
testcase tc_basedon_module_params(inout stats finalStats, charstring url, inout float totalTime) runs on client
{
	timer t:=20.0
	t.start;

	map(mtc:http, system:http);

	timer t_guard := 10.0;
	var float responseTime;
	var integer success := 0;
	var integer err404 := 0;
	var charstring host := "esekant027.epk.ericsson.se";
	var integer portNo := 80;

	var HeaderLines headerFields := {"Host: 136.224.74.26", "Connection: close"};

	var Close c := {};
	var Message t3rec;

	http.send(conn(host,portNo));
	http.send(get_ttcn_site(requestEsekant(headerFields,url)));
	t_guard.start;
	alt{
	[ ] http.receive(Message:rec_ttcn_site(rec)) -> value t3rec{
		responseTime := t_guard.read;
		t_guard.stop;
		setverdict(pass);
		success := 1;
	}
	[ ] t_guard.timeout{
		setverdict(fail);
	}
	[ ] http.receive(Message:rec_ttcn_site(badRec)) ->value t3rec{
		err404 := 1;
		http.send(Close:c);
		responseTime := t_guard.read;
```

```
		t_guard.stop;
		setverdict(fail);
}
[ ] http.receive {
		http.send(Close:c);
		responseTime := t_guard.read;
		t_guard.stop;
		setverdict(fail);
}
};


unmap(mtc:http, system:http);

if(success){
		finalStats.totalPages := finalStats.totalPages + 1;
		var charstring contentType := ""
		var headChar responseHeader := t3rec.response.header;
		var charstring contentLength := "";
		var charstring cache := headerParsing(responseHeader,contentType,contentLength);
		var charstring body := "";

		var integer bodylen := 0;

		//If body is present in the response, assign it to variable 'body'.
		if(ispresent(t3rec.response.body)){
			body := t3rec.response.body;
		}
		//If body is not present,take the length from 'contentLength' header
		//field. This is done for resources which are not of type text/html
		//The body is not fetched when the resource is other than text/html
		if(lengthof(body) > 1){
			bodylen := lengthof(body);
		}else{
			bodylen := str2int(contentLength);
		}

		//var charstring body := t3rec.response.body;
		//var integer bodylen := lengthof(body);

		var headChar urlList := {};

		if(cache == "private"){
			finalStats.privateCacheCnt := finalStats.privateCacheCnt +1;
		}else if(cache == "no-cache"){
			finalStats.nonCacheCnt := finalStats.nonCacheCnt +1;
		}else if(cache == "cacheble"){
			finalStats.cacheCnt := finalStats.cacheCnt +1;
		}

		if(contentType == "text/html"){
			finalStats.contentType.textHtml := finalStats.contentType.textHtml + 1;
		}else if(substr(contentType,0,5) == "image"){
			finalStats.contentType.image := finalStats.contentType.image +1;
		}else {
			finalStats.contentType.other := finalStats.contentType.other +1;
```

```
                }

                finalStats.totalSize := finalStats.totalSize + bodylen;

                testCaseReport(bodylen,contentType,cache,responseTime,url);
        }else{
                //If 404 error received,produce a report
                if(err404 == 1){
                        var charstring statusText := t3rec.response.statusText;
                        var integer statusCode := t3rec.response.statusCode;
                        testCaseFailReport(url,statusCode,statusText,responseTime);
                        finalStats.errorPages := finalStats.errorPages + 1;
                }else{
                        //If error other than 404
                        var charstring statusText := "Error other than 'Object Not Found'";
                        var integer statusCode := null;
                        testCaseFailReport(url,statusCode,statusText,responseTime);
                        finalStats.errorPages := finalStats.errorPages + 1;
                }
        }
        totalTime := totalTime + t.read;
        t.stop;
}



control
{
        var stats finalStats := {
                totalPages := 0,
                totalSize := 0,
                cacheCnt := 0,
                privateCacheCnt := 0,
                nonCacheCnt :=0,
                contentType := {
                        textHtml := 0,
                        media := 0,
                        image := 0,
                        other := 0
                },
                errorPages := 0
        }

        var integer len := sizeof(url);
        var charstring directory := "http://esekant027.epk.ericsson.se/042/";
        //      var charstring directory := "http://ttcn.ericsson.se/";
        //var charstring directory := "http://esekant027.epk.ericsson.se/042/General%20TCM%20Information/";
        //var charstring directory := "http://esekant027.epk.ericsson.se/019/";

        var float totalTime := 0.0;
        //To fetch one resource, use this test case
        //execute(simple());

        //Automatic WebSite testing
        execute(tc_static_check(finalStats,totalTime,directory));
```

```
        //Resource fetching based on the URL List provided as module params.
        // for (var integer i := 0; i < len ; i := i+1){
        //execute(tc_basedon_module_params(finalStats,url[i],totalTime));
        //}

        finalReport(finalStats,totalTime);



}

}

#include "HtmlParser.hh"

//This funciton parses the html code
//and returns urlList and number of outgoing links
//Parameters :
//    body    : htmlbody to be parsed
//    hostName: hostName
//    urlList : list of resrouces in the htmlbody
//    directory        : used to generate a proper path name for the urls
//    startingDirectory : used to generate a proper path name for the urls
//            example :../abc.html will get changed to http://ab.com/abc.html
void htmlParser(CHARSTRING body,
                                        CHARSTRING hostName,
                                        RECORD_OF<CHARSTRING>& urlList,
                                        CHARSTRING directory,
                                        CHARSTRING startingDirectory,
                                        INTEGER& outLinks)
{
        const char* htmlcode = body;

        char* s = (char*) htmlcode;
        int in_tag;

        char* dest = s;
        char* src = s;


        //Transform all separators into one space character
        while(src[0]){
                //add a space after '>' , the space will allow the later parsing
                //process function better
                if(src[0] == '>')
                {
                        dest[0] = '>';
                        dest++;
                        dest[0] = ' ';
                        dest++;
                        src++;
                }
                //carriage return = 13 , space = ' '
                //tab = 9 line = 10
                else if((src[0] == ' ') || (src[0] == 9) || (src[0] == 10)
                  || (src[0] == 13))
```

```
		{
			dest[0] = ' ';
			dest++;
			//eat up subsequent multiple occurances
			while((src[0] == ' ') || (src[0] == 9) ||
				(src[0] == 10) || (src[0] == 13)) {
				src++;
			}
		} else {
			dest[0] = src[0];
			src++;
			dest++;
		}
	}
	dest[0] = 0;

	//<img ...>, <area ...> and <a href ...> tags are kept, rest ignored.
	src = s;
	dest = s;
	while(strlen(src) > 8) {
		if(	(strncasecmp(src,"<a href",7) == 0) ||
			(strncasecmp(src,"<area ",6) == 0) ||
			(strncasecmp(src,"<img ",5) == 0)) {
			while((src[0]) && (src[0] != '>')) {
				dest[0] = src[0];
				src++;
				dest++;
			}
			if(src[0] == '>') {
				dest[0] = src[0];
				src++;
				dest++;
			} else src--;
		}
		src++;
	}
	dest[0] = 0;



  //Only urls are kept now.Everything else is removed. Example:
 // <img src="../images/foot_rcorner.gif" alt="foot_rcorner.gif (68 bytes)"
//WIDTH="10" HEIGHT="27">
	//will become : "../images/foot_rcorner.gif

	src = s;
	dest = s;
	while(strlen(src) > 8) {
		in_tag = 1;
		if(strncasecmp(src,"<a href",6) == 0) {
			outLinks = outLinks + 1;

			while(src[0] != '=')
			{
				src++;
			}
```

```
		src++;
		while(src[0] == ' '){
			src++;
		}

	} else if(strncasecmp(src,"<area ",6) == 0) {
		while(strncasecmp(src,"href",4)){
			src++;
		}
		while(src[0] != '='){
			src++;
		}
		src++;
		while(src[0] == ' '){
			src++;
		}
	} else if(strncasecmp(src,"<img ",5) == 0) {
		while(strncasecmp(src," src",4)) {
			src++;
		}
		while(src[0] != '='){
			src++;
		}
		src++;
		while(src[0] == ' '){
			src++;
		}

	}
	dest[0] = "";
	dest++;
	if(src[0] == '\'') {
		src++;
		while((src[0] != '\'') && (src[0] != '#') && (src[0] != '>')) {
			dest[0] = src[0];
			src++;
			dest++;
		}
	} else if(src[0] == '"') {
		src++;
		while((src[0] != '"') && (src[0] != '#') && (src[0] != '>')) {
			dest[0] = src[0];
			src++;
			dest++;
		}
	} else {
		while((src[0] != ' ') && (src[0] != '#') && (src[0] != '>')) {
			dest[0] = src[0];
			src++;
			dest++;
		}
	}
	while((src[0]) && (src[0] != '>')){
		src++;
	}
	while((src[0]) && (src[0] != '<')) {
```

```
                src++;
        }

}
dest[0] = 0;

src = s;

int urlNum = 0;
CHARSTRING url = "";
RECORD_OF<CHARSTRING> list = NULL_VALUE;
//The src is now a list of url in this format :"a1.html"b1.html"..
//put the urls in a list.
while(src[0]){
        if(src[0] == ""){
                src++;
                while(src[0] && (src[0] != "")){
                        url = url + src[0];
                        src++;
                }
                list[urlNum] = url;
                url = "";
                urlNum++;
        }
}

hostName = "http://"+hostName;
//update the urls with the directory.
//ex: a1.html will get changed to http://host/a1.html
getDirectoryLevel(list,directory,startingDirectory);

//If a url name has '&amp;', change it to '&'
for(int i = 0; i<list.size_of();i++){
        if(strstr(list[i],"&amp;") != NULL){
                CHARSTRING url = "";
                const char* link = list[i];
                char* s = (char*)link;
                char* src = s;
                while(src[0]){
                        if(strncasecmp(src,"&amp;",5) == 0){
                                url = url + src[0];
                                src++;
                        src = src+4;
                        while(src[0]){
                                url = url + src[0];
                                src++;
                        }
                        }else{
                                url = url + src[0];
                                src++;
                        }
                }
                list[i] = url;
        }
}
```

```
	CHARSTRING n1 = "";
	int k = 0;

	//Remove duplicate values from the array.
	for(int i = 0; i<list.size_of();i++){
		n1 = list[i];
		//remove elements which are not in this host
		if(strncasecmp(n1,hostName,hostName.lengthof()) == 0){
			//		cout << "n1 " << n1 << endl;
			for(int j= 0; j<list.size_of();j++){
				int found = 0;
				int f = 0;

				if(n1 == list[j]){
					for(int i=0; i<urlList.size_of();i++){
						if(n1 == urlList[i]){
							f = 1;
							break;
						}
					}
					if(f)break;
					urlList[k] = n1;
					k++;
					found = 1;
				}
				if(found){
					break;
				}

			}
		}

		n1 = "";
	}

}

void getDirectoryLevel(RECORD_OF<CHARSTRING>&list,CHARSTRING directory,CHARSTRING
startingDirectory)
{
	//directory can contain '\0', const char* creates a null terminated string
	const char* tmp = directory;
	CHARSTRING dir = tmp;

	for(int i=0; i < list.size_of();i++){
		int appendHost = 0;
		CHARSTRING url = "";
		CHARSTRING tmpDir = "";
		//if list[i] has a url : ../ab.html
		if((strncasecmp(list[i],"../",3) ==0)){
			tmpDir = dir;
			int len = strlen(tmpDir);
			tmpDir[len-1] = '\0';
```

```
		len = strlen(tmpDir);
		for(int i=len; i !=0;i--){
			if(tmpDir[i-2] == '/'){
				tmpDir[i-1] = '\0';
				break;
			}
			tmpDir[i-1] = '\0';
		}
		url = url + (const char*)tmpDir;
		appendHost = 1;
}else if(strncasecmp(list[i],"./",2) == 0){
		url = url + dir ;
		appendHost = 1;
}else if(strncasecmp(list[i],"/",1) == 0){
		url = url + startingDirectory+ list[i];
}else if(strncasecmp(list[i],"file",4) == 0){
		url = url + list[i];
}else if(strncasecmp(list[i],"mailto",6) == 0){
		url = url + list[i];
}
else if((strncasecmp(list[i],"../",3) != 0) &&
			(strncasecmp(list[i],"./",2) != 0) &&
			(strncasecmp(list[i],"http",4) != 0)){
		url = url + dir + list[i];
}

const char* link = list[i];
char* s = (char*)link;
char* src = s;

if(appendHost){
		while(src[0]){
			if(strncasecmp(src,"../",3) == 0){
				src = src+3;
				while(src[0]){
					url = url+src[0];
					src++;
				}
			}
			else if((strncasecmp(src,"./",2) == 0)){
				while((src[0])){
					url = url + src[0];
					src++;
				}
			}
			else if((strncasecmp(src,"../",3) != 0) ||
					(strncasecmp(src,"./",2) != 0) ||
					(strncasecmp(src,"http",4) != 0)){
				url = url + src[0];
				src++;
			}
			else{
				src++;
			}
		}
}
```

```
                list[i] = url;

        }

}

#include <iostream.h>
#include <stdio.h>
#include <TTCN3.hh>
#include "HTTPTest.hh"
#include "HTTPTestCases.hh"

void htmlParser(CHARSTRING body,
                                CHARSTRING hostName,
                                RECORD_OF<CHARSTRING>& urlList,
                                CHARSTRING directory,
                                CHARSTRING startingDirectory,
                                INTEGER& outLinks);

void getDirectoryLevel(RECORD_OF<CHARSTRING>& list,
                                        CHARSTRING directory,
                                        CHARSTRING startingDirectory);
```