Ensuring Quality of Web Applications by Client-side Testing Using TTCN-3

Cosmin Rentea[1], Ina Schieferdecker[1], and Valentin Cristea[2]

[1] Fraunhofer FOKUS Institute, Berlin
{ cosmin.rentea, ina.schieferdecker }@fokus.fraunhofer.de
[2] Politehnica University Bucharest
valentin.cristea@cs.pub.ro

Abstract. Although there are many test systems and languages which were designed and tailored specifically for the Web, the versatility of standardized test languages makes them good candidates for such testing activities. TTCN-3 is a standardized test language and test architecture for distributed reactive black-box testing. In this paper, we present a framework for testing Web applications and Web services using TTCN-3 as language of choice. Besides this abstract test layer, we have developed a test adapter and codec that permit seamless integration and usage of browser simulators while providing for functional testing as well as non-functional testing (e.g. distributed load-testing, security testing). The capabilities were proved for certain proof-of-concept web applications; a case study illustrates the relevance of the framework in the context of quality assessment and testing.

1 Introduction

The purpose of work underlying this article is allowing a rich variety of testing methods using TTCN-3 and applied to Web applications.

Over the last years, there was a spectacular growth of the web applications domain. Dynamic content generation and scripting languages (server-side like PHP, and client-side as JavaScript) are used to develop these applications, leading to heterogeneous interactions between the components of the application, which are aspects of a very error-prone development process. Rigorous testing must be enforced, usually following the multi-tiered approach of the web application under test.

At the same time, the content formats of the web documents are constantly changing. The appearance of more semistructured XML-conformant languages like XHTML and information retrieval and tranformation languages such as XPath, XQuery, XSLT, made the structural aspects of Web documents more important than the presentational ones. These formats (including SOAP for web services) are standardized by the W3C and supported by syntactic validators. This means the testing can focus now on the actual semi-structured content and information flows.

2 Context and Related work

2.1 TTCN-3 – Test Language, Architecture and Tools

TTCN-3 (Testing and Test Control Notation version 3) is an abstract testing language that can be used for black-box testing of any reactive system, with any communication interface. It provides a way of describing so called Abstract Test Suites (ATS), that are independent of the test execution environment. TTCN-3 [1] is the only standardized language for the specification of test cases.

TTCN-3 allows easy and efficient description of complex distributed test behavior. It contains all the features necessary to specify test procedures and campaigns. In addition to the pure textual format, the Tabular and Graphical Presentation Formats have been defined to provide graphical means both for data and behavior aspects. New concepts are test behavior in asynchronous and synchronous communication environments, test component concepts to describe concurrent test setups, dynamic concurrent test configurations, integration of data-definition languages (ASN.1, IDL, XML Schema).

The top-level unit of TTCN-3 is a module, which can import definitions from other modules. Modules consists of a definitions part and a control part, and can have parameter lists. The control part of a module calls the test cases and controls their execution (sequences, repetitions and dependencies on test outcomes). The definitions part of a module (which can be split into groups) defines test components, communication ports, data types, constants, test data templates, functions, signatures for procedure calls at ports, test cases etc. A collection of test cases is known as a test suite, its execution being a test campaign.

TTCN-3 has a similar look and feel to a typical programming language. Program statements can be used to specify the selection and execution order of individual test cases. TTCN-3 has a number of predefined basic

data types, and structured types such as records, sets, unions, enumerated types and arrays. Data structures called templates provide parametrization and matching mechanisms for specifying test data to be sent or received over the test ports. The operations on these ports provide both message-based and procedure-based communication capabilities. TTCN-3 program statements include behavior description mechanisms such as alternative reception of communication and timer events, interleaving and default behavior. Multiple components can run their own test logic in cooperation with the others and give their local verdict to establish the final global verdict of the testcase. Test verdict assignment and logging mechanisms are also supported. Finally, TTCN-3 language elements may be assigned attributes (even user-defined) such as encoding information and display attributes.

A TTCN-3 test system can be conceptually thought of as a set of interacting entities. The part of the test system which implements interpretation or execution of a TTCN-3 test specification is represented by the TTCN-3 Executable (TE) entity. This entity corresponds either to an interpreted or an executable test suite (ETS) produced by a compiler from a TTCN-3 ATS. The remaining parts of the TTCN-3 test system, which deal with any aspects that can not be concluded from information present in the original ATS alone, can be decomposed into Test Management (TM), Component Handling (CH), Codec (CD), SUT Adapter (SA), and Platform Adapter (PA) entities. As depicted in Fig. 1, a TTCN-3 test system has two major interfaces, the TTCN-3 Control Interface (TCI) and the TTCN-3 Runtime Interface (TRI), which specify the interface between Test Control TC (comprising TM, CH and CD) and TE entities, and TE and Test Runtime TR (SA + PA) entities, respectively. More information about the language can be found on the official webpage [1], while a good introduction is given in [2].

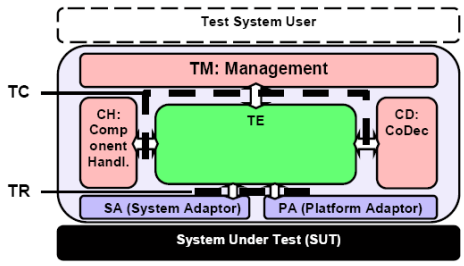

Fig. 1. TTCN-3 System Architecture

TTCN-3 can test all types of reactive system over a variety of communication ports. Typical areas of application are protocol testing, services testing, module and API testing, system and middleware testing (e.g. CORBA, EJB or web services technologies). TTCN-3 is not restricted to conformance testing and can be used for many other kinds of testing including interoperability, robustness, regression, system and integration testing.

TTworkbench is a graphical test development and execution environment for TTCN-3, being implemented in Java on top of Eclipse as a series of plugins that add new perspectives: one for specificaiton of TTCN-3 modules and testcases; one for management, execution and analysis of executable testcases. Once tests have been defined in TTCN-3, TTworkbench compiles them in an ETS using the TTthree compiler (first Java classes are generated, then they are compiled using the usual Java compiler).

2.2 Related Work using TTCN-3 Approaches

We reviewed below the contributions that employ TTCN-3 technologies for enabling testing of web applications and web services.

The authors of [3] proposed a TTCN-3 based architecture for complex web-testing; however, it is just presented as a theoretical approach. The components of their testing methodology are: (1) multiple test methods are employed for different models: analysis model, (e.g. use case, object state diagram) and design model (navigation model, business model, data model, sequence diagrams etc.); (2) TTCN-3 is used to specify all the testcases on an abstract level (ATS) – this helps separate test design from test implementation and makes ATS language and platform independent, increases the reusability of TCs and the degree of automated testing, takes advantage of the commercial tool support enjoyed by TTCN-3; (3) Several other test tools may be used such as link checkers, and GUI capture-and-playback tools; TTCN-3 translators may be required to convert ATS to the proprietary scripting languages of test tools.

A life-cycle testing process for e-commerce systems by adapting OO-TTCN-3 (an object-oriented extension of a formal test language TTCN-3) to enable the efficient specification of tests in object-oriented, e-commerce development environments, is presented in [4]. This extension is meant to ease life-cycle testing, facilitate test case reuse between different test phases, and provide a unified ATS interface to test tools. The OO-related extensions are inheritance hierarchies (extend and override mechanisms, usage of *private* and *public* reserved words) and aggregation relationships between test modules. The author implements only a very basic and limited test adapter and codec. This is only a theoretical approach, as these extensions are not implemented in any TTCN-3 tools. Overall, this is does not leverage TTCN-3 – the presented test suite can only be used for verifying that a URL can be loaded.

In [5] the authors employ TTCN-3 to define test cases at different levels of abstraction that are more robust in the face of volatile presentation and implementation details. A case study of a shopping cart scenario with order processing is used as an illustration. Features of TTCN-3 are demonstrated, including a powerful matching mechanism that allows a separation between behavior and the conditions governing behavior; TTCN-3s data types and set-based operations allow one to track and verify the information management done by a web application, independent of implementation details. The advantages and challenges of a systemic test specification approach are characterized in comparison to approaches based on unit testing and test automation tools. The main differences between the proposed approach and the is the emphasis put on reutilisation and automation of TTCN-3 definitions and templates, the aggregation of XPath to check not only navigational issues but also structural ones, the increadesd complexity of TTCN-3 Adaptation Layer, and the usage of configurable, complex, visual tools both during identification of test objectives and test execution.

In [6] a TTCN-3 abstract testing approach is proposed for web services, which distributes the test activities to both server and client sides, thus facilitating testing while the traditional approaches fail due to the difficulties brought by the language and platform-independence of web services. Nevertheless, this approach lacks generality as it is tailored toward specific web services even in the abstract test suite, and the adaptation layer can hardly be adapted to new real-world services. The binding between web service definition and the TTCN-3 system is not automated after the test case design phase.

An interesting contribution to the domain of testing web services is done in [7]. The paper discusses the automated testing of Web services by use of TTCN-3. A mapping between XML data descriptions to TTCN-3 data is presented to enable the automated derivation of test data and test configuration. This is the basis for functional and load tests of XML interfaces in TTCN-3. The paper describes also theoretically the mapping rules and prototypical tools (an XML to TTCN-3 conversion tool, a test adapter for XML/SOAP interfaces, and the compiler) for the development and execution of TTCN-3 tests for XML/SOAP based Web services.

In the same line of generating partial TTCN-3 definitions (data types, templates, even simple behaviour in the form of stubs) in a generative process starting from XML specifications following W3C web standards, we could cite the work done by [8] for XML Schema, and by [9], [10] and [11] for WSDL.

The principles of distributed testing using the TTCN-3 TCI are described in [12]. In a similar perspective applied to the web services domain, [13] describes an automated TTCN-3 test framework whose purpose is easing the creation and deployment of distributed functional and load tests. The test framework is exemplified for web service tests and presents results obtained from testing a specific web service.

3 Requirements, Design and Architecture

We want to allow the testing of normal web applications using *reactive black-box* mechanisms.

What matters is not the technology that is used on the server-side for page generation, but satisfying the project's requirements from the user's point of view – thus, the perspective of "client-side testing" was adopted. We want to analyze only the actual content received at the client from the Web servers and see if they correspond to our expectations; such content includes HTML/XHTML documents, images, JavaScript embedded in documents or in separate files, even CSS files. The aim is to be able to simulate the behavior of a browser as accurately as possible. Active components (like Java applets, Flash objects) are excluded from testing because communication between them and the server can be arbitrary using TCP/IP sockets; interpretation of these interactions is application-specific, so we'll test only standardized information passed at the HTTP protocol level.

A desired requirements is to have not only navigational checks, but also document and content testing. To achieve this, we use either XPath or XQuery to analyze the structure of the received documents. These are the established standards that provide the possibility to access fragments of the XML-based documents (like XHTML) by using a common expression algebra (based on XML Information Set Data Model). Since these languages contain small textual expressions, we can maintain for each webpage we test a list of queries that identify respectively the portions we want to check. The results of these queries are in the most general

case sets of nodes (e.g. one row in an HTML table), but can also be single nodes or strings or numeric results. For instance, one can check the presence of all the required fields in a form by using different queries to select all the controls (e.g. checking for correct captions or associated actions for buttons, searching for a specific text etc.). The list of query expressions specified above can be stored in an external repository along with the identifiers for applications and for current webpages. These identifiers can take into consideration the URL of the document, its parameters and the history of previously visited pages.

Another goal is to provide the tester with multiple test approaches. Testing of web applications may be either functional (testing specified use-cases and scenarios), conformance testing (for the application specifications and validators for documents, links, objects) stress and performance (simulating multiple clients, time/load measuring), security. Finally, we want to provide realistic implementation usable for system and integration and regression testing.

We wanted to provide an open standards-based testing infrastructure for web-applications. TTCN-3 was selected as the testing language of choice because of its features: a standardized, abstract test-language designed for black-box reactive testing and having a clearly-defined interaction with the SUT. Having defined our requirements, the next step was to identify the usable standards, software and tools. A decision was made to use TTworkbench as the development environment; the arguments were the features, extendability, development time and quality of support.

The main design components are the following:

Mapping Generator (WebTestGUI) for the identification of parts to be checked on webpages.
Mapping Repository that store pointers or constraints related to various elements on the different pages of the web application.
TTCN-3 Web API is a high-level API oriented to Web-testing that defines data types, functions, signatures, components.
Test Adaptation Layer comprises the parts of the architecture that are residing at the TRI [14] and TCI level [15]: Test Codec (TCI, CoDec interface); Test Adapter (TRI, System Adapter interface); Test External Functions (TRI, Platform Adapter interface). The TTCN-3 API and the Test Adaptation are collectively named TWeb.

The advantages of this design are conformance to standards (W3C and ETSI), generality (applicable to every XML-based document), separation of concerns (between identification of testable items and actual testing, and between the different levels of test abstraction), loose coupling between the software components (by using a repository), automation (permits the generation of executable tests starting from the repository), configurability (settings on Adaptation layer); On the source code level, maintainability, reusability and extensibility are assured: TTCN-3 code and definitions are modularized, new TTCN-3 templates can be derived, Mapping Repository generates and reuses similar regular expressions for applications webpages, codec and adapter are reusable and extendable.

3.1 Mapping Repository and Mapping Generator – WebTestGUI

Each *Repository* consists of *Pages* which are defined by a set of *Addresses* which can have multiple *URLs* (specified as *regular expressions*) and HTML *parameters* (*name=value* pairs). It can be also possible to specify a browsing *History* (the ordered list of pages that had to be visited first before "activating" the current entry).

To specify the parts we want to test on the page, we defined the *Mapping* structure. Each *Address* represents a page loaded in the browser or by the test adapter; it can be specified at different moments in the web-application workflow and with varying parameters or under different URLs. Overall, a *Test* contains multiple *Adresses* with *regular expresion URLs* and with multiple *XPathMappings* that contain *Name=XPathExpression* associations. The structure of the Mapping repository is described in Fig. 2. This hides from the TTCN-3 tester the details of the underlying technology used for accessing nodes in the XML document (XPath or XQuery); the programmer only needs to specify the given name of a query and if the *Mapping* is active in the moment of the call (by searching through the list of activated *Addresses*) then the test adapter will evaluate the expression and return the results. The programmer needs to indicate to the test adapter which repository will be used in his testcase and the test adapter will do the rest. In this way, we can have a great deal of control over the *Mappings* that are active at a certain time; it is easy to check the static or the dynamic portions of HTML at runtime because different *Addresses* will be active depending on the parameters and browsing history. Also, we can make tests to the same application deployed to different hosts, ports etc. by using the regular expression mechanism when specifying the URLs that can activate an *Address*.

Taking into consideration that the structure is hierarchical, the mapping repository can be easily specified as an XML file, whose structure can be defined and validated using XML-Schema. After experimenting with

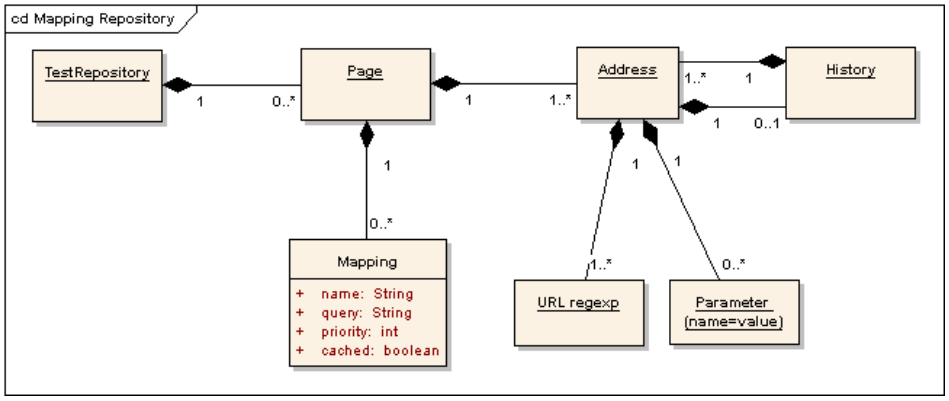

Fig. 2. Mapping Repository Structure

both standards and related tools, we took the decision to use XPath and not XQuery because (1) XPath is easier to understand and program with, while XQuery is a recent standard; (2) XPath expressions can be automatically generated (to a certain degree) and are just strings and can be easily kept inside XML files, while XQuery expressions are more complicated and would need to be kept separately in different files in a hierarchical structure (which is harder to maintain); (3) Only the XPath addressing was compulsory, while the complex expression of XQuery could be added at a later stage.

The purpose of the WebTestGUI tool is to obtain Mapping repositories. The software must be able to export a XML file containing the Mapping repository; this is done by the MappingWriter component, which is contained by the main component. The selected BrowserSimulator will be a package that implements DOM-Events Level 2 [3]. Users will use the GUI which contains also the BrowserSimulator canvas in order to visually select elements on the page. When selecting such a node (for instance by clicking a hyperlink or entering a textfield), a DOM-Event will be fired and then handled by the DOM-Events Handlers, and information about the Node will be analyzed and displayed in the GUI according to its type. In order to permit browsing and mapping selection, the program needs to work in two modes:

1. *Interception*: The DOM-Events are collectively intercepted. Nodes on the page can be interactively selected and added to the repository. Every navigation action is canceled (for instance clicking on links has no effect because after intercepting the DOM-Event, the event is discarded).
2. *Navigation:* Permits browsing as in a normal browser.

These modes can be selected at any time by a control in the GUI. The functional regime is the following: after loading a new page (*navigation*), the tool will switch automatically to *interception* mode; after the user has added to the repository all the mappings relevant for the current page, he must change the mode back to *navigation* and continue browsing.

3.2 TWeb – Integrated Web Testing Solution with TTCN-3

Web Testing API in TTCN-3 There is unfortunately no standardized notation for expressing TTCN-3 modules (like UML Class Diagrams) because TTCN-3 is not object-oriented (although it is module-oriented). Also, because we are not describing a process, but a set of functions, signatures and structures, we cannot use activity diagrams for description of the behavior.

Nevertheless, we can describe the intentions of the modular design. The API in located inside a single TTCN-3 module, which is structured into several groups, i.e. TTCN-3 convenience functions, definitions and functions for handling URLs, data structures, signatures and functions for handling HTML pages and specific content, data structures, signatures and functions for handling HTTP connections, data structures, signatures, functions that embed a Browser, definitions of procedure-type ports and of components that embed them.

Conceptually, the *Browser* can create and handle multiple *Windows* that contain stacks of *WebPages*. Each *Window* can have its specific properties (like using a proxy, disabling JavaScript or object-loading). Only one *Window* and one *WebPage* is active at every moment in time; the possibility exists to go back and forward in the chain of *WebPages* in each *Window*. A *WebPage* contains in turn a *HttpRequest* and a *HttpResponse* structure.

[3] http://www.w3.org/TR/DOM-Level-2-Events/

The API logic is implemented by three means: external functions (implemented in the Platform Adapter), signatures (implemented in the SUT Adapter), and functions.

External functions are functions implemented in Java and called in TTCN-3 for features that

- are already well-done in Java and difficult to re-implement in TTCN-3, for instance handling *URLs* with the standard Java *URI* class, or more complex logging
- can only be done in Java: issuing database commands (e.g. for reloading the same data in the database to assure consistent results, not influenced by any possible previous testcase failures), conversion functions, accessing the execution platform

Signatures are used for things that are related to the communication (HTTP requests and responses, browser settings) and the page analysis (DOM-handling, evaluation of currently active Mappings, specific functionality for HTML Tables and Forms etc).

TTCN-3 functions are used to implement associative arrays, analyze the HTTP headers, set the HTML parameters, analyze the DOM-tree in TTCN-3, for initialization of the browsing session parameters (host, port etc.), interacting with the page controls and forms and for getting any element(s) from the page, as well as for testing HTML Tables.

TWeb Adaptation Layer comprises three components:

Test Codec - TCI, implementing the *CoDec* interface)
Test External Functions - TRI, implementing a part of the *Platform Adapter* interfaces
Test Adapter - TRI, implementing the *SUT Adapter* interface

These components are also described in a UML Component Diagram (Fig. 3). They are implemented in Java as this is the language provided by TTworkbench platform for writing adapters and codecs.

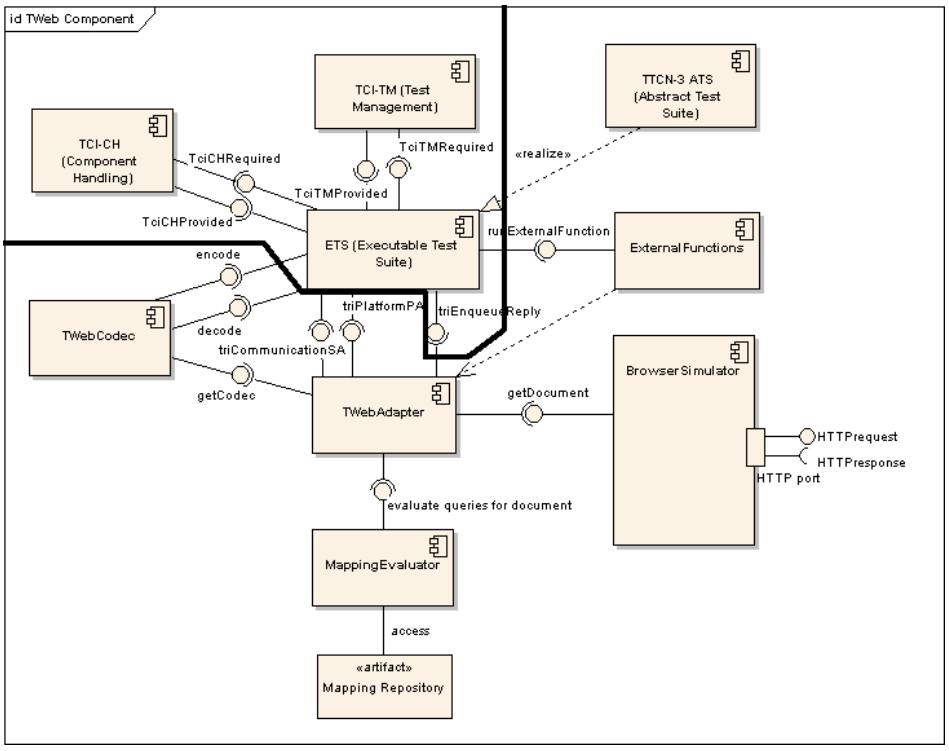

Fig. 3. TWeb Component Diagram

Encoding functionality has to be provided to the TTCN-3 runtime behavior in order to send real data over a real communication port, or call a real world method on a SUT port. On the other hand incoming data has to be decoded into a TTCN-3 value, to allow matching against TTCN-3 template definitions [15]. Two entities have to be implemented: an *Encoder* (TTCN-3 data to real world data) and a *Decoder* (real world data to TTCN-3 data); the new implemented Encoder and Decoder have to be announced to the runtime behavior using the methods provided by the TCI Interface. Two interfaces are provided by the platform:

– *Interface "Encoder"* that offers the encode method, that takes as a parameter a *Value* value and returns a *TciMessage*, encoded message. There is a basic encoder provided by the underlying platform that encodes all TTCN-3 data into a byte array.
– *Interface "Decoder"* that offers the decode method, that takes as parameters, a *TciMessage* and a *Type*, and decodes the message, and the returned *Value.type* contains the expected type of the decoded message (decoding hypothesis).

The underlying TTCN-3 platform provides a useful extension for writing external functions by respecting rules about parameter-access and implementing a generic interface.

The TWeb Adapter includes a *BrowserSimulator*. A choice had to be made according to the features of the software tools (e.g. JRex, HtmlUnit[4], or Lobo[5]). Although a single tool was selected, it is possible as well to use the other packages as browser simulators by rewriting parts of the Adapter.

In the adapter there is an ad-hoc procedure for handling of the incoming *triCall()* (corresponding to the port *call* in TTCN-3). Each signature is implemented in its own Java method, receiving its parameters (*TriParameter*). Returning the results is done only in *triCall* for all the signatures.

The *Mapping Loader* component reads at first the *Mapping* entries from the repository and then when the TTCN-3 user changes the page loaded in the *BrowserSimulator* (by using for instance the *load()* signature), it changes the set of active *Mappings* depending on the new URL (matched against a set of regular expressions), parameters and browsing history. Then, if required, it evaluates this modified set of active *Mappings* using an XPath package; the class used to implement a *Mapping* (name, expression along with all its possible results) is *XPathMapping*.

4 Implementation

4.1 Software Packages used in the Implementation

Jaxen is an open source cross-API XPath library for Java[6]. It is a Java class library that can operate on various XML object models using a standard engine rather than an API that can be offered by many different engines for one model. Jaxen is a more complete object model for XPath expressions, especially for evaluating XPath expressions in a a particular document against a context node.

JRex is a Java-browser Component with set of APIs for embedding the Mozilla browser within a Java application[7]. It provides a Java wrapper around the required Mozilla embedding interfaces; it does not require Mozilla to be installed, instead uses GRE (Gecko Runtime Engine – a minimal browser engine); since Mozilla is composed of multiple interfaces (mostly in C++ and JavaScript), controlling the browser is simply a matter of accessing these interfaces from Java using Java Native Interface. It's compatible with the latest versions of Mozilla Gecko, can run on multiple platforms. It has an easily extendable APIs, permitting integration of other emerging W3C standards like XHTML 2, XForms etc. JRex is compatible with AWT, Swing and also with Eclipse's SWT. JRex supports tabbed and single-pane browser windows, profile and preferences (to enable/disable plug-in, images, setting web-proxies etc.), Persistence, History, ContextMenu, ContentUrlListener, Observer, Progress, window and event capturing and management. Most importantly, it provides native support for accessing DOM objects of rendered page. It implements partly DOM3 HTML for manipulating loaded HTML Document and also DOM-Events Level 2. The JRex DOM implementation has additional features: the browser can read and interpret correctly even malformed HTML, while the tags are always rewritten in an uniform way; the provided DOM is namespace-aware. JRex is similar to a real browser (e.g. runs the JavaScripts embedded in HTML) and it can be used for accurate user-oriented performance measuring (capacity, latencies etc.) – for instance, the load-time of the page is more accurate from the user's point of view, given that a browser is not only receiving some text on a HTTP connection, but also the rendering, retrieving other documents and objects etc.

The reasons to choose JRex were: using a real embedded browser, it offers a graphical view of the web pages; it is more realistic from the user's point of view for time measurements and responsiveness; it is extensible and configurable (through JNI); Mozilla/Seamonkey code-base has many functionalities, being also maintained and supported; it is a free open-source solution; it can be used in a dual role – also in the Mapping Generator tool to create Mapping Repositories.

Mozilla 1.8 was compiled using the Cygwin environment and the Microsoft C++ compiler in order to get a recent Gecko engine. Jrex contains these libraries as well as compilable Java classes that access them using JNI.

[4] http://htmlunit.sourceforge.net/
[5] http://lobobrowser.org/
[6] http://jaxen.org/
[7] http://jrex.mozdev.org/

4.2 WebTestGUI and the Mapping Repository

WebTestGUI is implemented in Java using JRex (integrated in a Swing application), Jaxen, JAXP.

JRex was used as a Java Canvas descendent, embedding it in Swing frames as we would with a normal component. The program needs to be notified when the browser finished loading a page, and after that it is needed to install the DOM-Events handlers. In the event handler, after finding the selected DOM Node and finding its type, the event is consumed because the browser should not actually handle it.

The GUI contains a list of the mappings, with the possibility to add, edit or delete them. Information shown includes data about the node, its attributes and children, contained text, suggested name of the Mapping, an automatically generated unique XPath expression identifying the set of selected nodes in the DOM, along with the evaluation of the current XPath expression. The user can edit the current mapping by changing its name and XPath expression (with real-time evaluation on the loaded DOM using Jaxen), or by selecting a different node from the DOM tree that is presented in the GUI. Page URLs can be also edited and tested in the context of the current page; regular expressions are accepted if they match the current URL. The tool can intercept any relevant DOM-Events related to visible elements (form controls, tables, text etc.) on the page, and gives visual feed-back on selection. It can also be used for selecting invisible elements by using the DOM-tree control in the GUI. Since DOM-Events are closely bound to the DOM structure of the page, loading a new page in the BrowserSimulator resets these handlers, which have to be installed again. The current Mapping Repository can be saved or loaded using the MappingWriter/MappingReader components, using standard Java JAXP package; these mappings can be later referenced in the TTCN-3 API by their given names. The definitions in the repository will be contained in an XML file. Each mapping entry will be placed according to the introduced URL regular expression.

4.3 TWeb

Web Testing Library Module in TTCN-3. HttpRequest can specify for instance HTTP method, the full URI, the protocol, the headers etc. HttpResponse is more complex and contains the loaded URI, the protocol, status code and string, the load-time (time necessary to load the page), headers and the raw content of the received stream, and a HtmlPage structure (which can be modified in case of other content type, e.g. regular XML files). The HtmlPage record has four types of fields: (1) a DOM-like tree structure, composed of structures describing Nodes; (2) current Mappings for the page; (3) the Controls of the forms on the page; (4) various properties that can be computed from the page; these fields, depending on preferences on the Browser level, can be computed either at the adapter level, after loading the page (default), or with signature calls after receiving the page in TTCN-3 (e.g. for performance measurements).

Also, in the HTTP and HTML groups types are defined types for handling HTTP headers, HTML parameters, HTML DOM Nodes, XPath Mappings and their evaluation results, all HTML controls and forms, HTML tables. Templates for HTTP response-matching, HTML page-matching, and HTML table-matching also were defined here; these can be derived to get more specific templates usable in specific testcases.

As an example, we describe two signature definitions, actually executed in the Adapter:

load takes an HttpRequest, passes it to the Adaptation layer, which loads the page in BrowserSimulator and evaluates the Mappings; the adapter issues back a HttpResponse which can be interpreted in the TTCN-3 code.

getFullHtmlTable takes an HtmlPage and the table index and returns an HtmlTable with the table dimensions and the cells as a matrix with HTML Nodes.

TTCN-3 functions are used in several places; they implement associative arrays, analyze the HTTP headers, set the HTML parameters, analyze the DOM-tree in TTCN-3, are used for initialization, interacting with the page controls and forms, and for getting any element(s) from the page, as well as for testing HTML Tables.

The definition of a port and a component were also defined; the port must be of *procedure* type, acting as in synchronous Remote Procedure Call:

```
type port HttpPort procedure{
out     createBrowser, createWindow, configureWindow, setActiveWindow, load, clickLink, clickButton,
        goForward, goBack, testObjects, testLinks, testConformance,     ..........
        getFullHtmlTable;
in      WebBrowser, WebWindow, integer, charstring, boolean, float, Map,
        HtmlNode, HtmlNodes, MappingContent, HttpResponse, HtmlTable; }
type component WebComponent { port HttpPort httpPort; }
```

It is possible to get the results of XPath Mappings evaluation using a function with an XPathMapping parameter (which contains its name), or to evaluate non-named XPath expressions (which are not already present in the Mappings Repository). One can search for regular expressions in the text nodes of the page

and to explore nodes, attributes for finding their properties (such as the URL of a link or the link in a table cell).

Nodes are represented by a HtmlNode structure which mimics a DOM Node by using a TTCN-3 record with optional fields. For now, the types of nodes received from the adapter can be XML elements, attributes or text; the usable fields are determined by the node type. Convenience functions are provided for handling these nodes.

Controls are modeled with TTCN-3 records with optional components following the HTML Forms[8] specification. All their possible attributes are specified, and according to their type they are handled differently by the convenience functions; for example, setting or resetting the value of text input compared to checkboxes have different meanings. The Controls can be converted back and forth from regular Nodes. The HtmlForms records can contain multiple controls; only one form can be submitted at one time, and then if applicable the values of the successful controls are transmitted as request parameters (either with the GET or POST method). Another important issue is handling HtmlTables; these are received from the adapter using a signature and are contained in a record which embeds information about its dimensions, headers, and the cells represented as TTCN-3 HtmlNodes. Some general tests are also included, e.g. for checking links integrity for the whole page, or checking for embedded objects.

The TTCN-3 Codec must implement the *TciCDProvided* (TCI) interface. There is an *AbstractBaseCodec* that handles the default encodings if necessary. The TTCN-3 language also provides means to specify the encoding of a structure/record. The Codec supports Unicode encodings for strings.

The following problem was encountered: the Java representations of TTCN-3 variables (*Value* clasees) should be accessed in the Adapter as they are transmitted from the Test Executable. These TTCN-3 values may come in structured forms like records, and should be passed as parameters to underlying software in the adapter (such as the Browser Simulator or the XPath engine). The Codecs are supposed to transform these *Values* into byte arrays, that should be again decoded again in the Adapter in order to interact with the SUT. This affects unfortunately many cases of API calls done in the Adapter with parameters from the TTCN-3 ATS.

Envisioned solutions for bypassing the Codec are (1) Serialization of *Value* classes representing the parameters; (2) Encoding Values as XML in the codec and parsing the stream in the Adapter; (3) Using Java introspection mechanisms to transmit values from TTCN-3 Values representation into Java classes generated when compiling the TTCN-3 modules; (4) A hashmap with auto-generated string keys and TTCN-3 *Values* – transmitted or created Values are put into and taken from this structure. The latter was implemented in practice, so *Values* are handled almost entirely in the Adapter.

External Functions were defined in a Java class that implements the functions defined as *external* in the TTCN-3 ATS. Such functions are implemented for: *Execution of SQL scripts:* connection to a database through JDBC and execution of arbitrary commands – used for resetting the data in a database; *URL:* handling and conversion functions are already implemented in Java; *Logging:* logging of complex objects is done by a Java logging package rather than the TTCN-3 log() statement.

The TTCN-3 TWebAdapter is derived from *TestAdapter*, which is a base class provided by the platform to fulfill basic needs (and extends Thread implementing the standard TriCommunicationSA, TriPlatformPA, TciEncoding).

We used a RPC approach by using TTCN-3 signature calls. *Calling* at the TTCN-3 is reflected in the Java *triCall* being invoked. Inside this function, the signatures are dispatched to their Java corresponding methods – each signature is implemented in a method (the TTCN-3 standard exception and reporting mechanisms were enhanced to allow this).

The selected BrowserSimulator was JRex, as in the case of the mapping-generator tool. A feature provided by the adapter is "exporting" the DOM tree of the loaded HTML file obtained from JRex into the corresponding structures of the TTCN-3 API.

Inside the Adapter, the MappingLoader reads at first the Mapping entries from the XML repository. Then, when the TTCN-3 user changes the page loaded in the BrowserSimulator (by using for instance the load() signature), it changes the set of active Mappings depending on the new URL (matched against a set of regular expressions), parameters and browsing history. Then it evaluates this modified set of active Mappings using an XPath package (Jaxen).

Another feature provided by the adapter is "exporting" the DOM tree of the loaded HTML file into the corresponding structures specified in the TTCN-3 API; this is done in TWebAdapter by generating nested records of TTCN-3 types HtmlNode. This method can be called with the root *Node* of the Document as a

[8] http://www.w3.org/TR/html5/forms.html

parameter and then the whole tree is received in a TTCN-3 variable, or with a regular Node as a parameter after the MappingLoader generates MappingResults that represent a single Node or a set of Nodes.

In the implementation we focused on HTML Controls, Forms and Tables because these are very common among the web-applications of our days, which generate dynamic-content following user-interaction. Handling of HTML tables is enabled through XPath expressions for selecting the table in the page by its index, computing its dimensions and providing the content of the cells as HtmlNode structures. The HtmlNodes are always exported along with all their descendant nodes, attributes etc. Handling of various HTML controls is done similarly to tables.

The adapter does also timing measurements that accurately indicate the time necessary for fully loading a page and rendering it. Many important features supported by normal browsers are implemented in the TWebAdapter, such as (1) Cookie handling; (2) HTTP parameters encoding for GET and POST methods; (3) Scripting is taken into consideration, JavaScript is enabled but can be programatically disabled; (4) Web proxies are supported, also by means of Browser/Window properties; (5) Relative URLs are transformed into absolute URLs; (6) Automatic redirection happens transparently for the TTCN-3 in the case of HTTP or HTML redirection: the HTTP redirect by analyzing the connection status code and loading the page specified in the HTTP header; the HTML redirect by analyzing the page and interpreting it like a normal browser (taking into consideration also the timing).

5 Case studies and Results

BookSearch. Both WebTestGUI application and TWeb testing system were successfully used in a project having as client a major Japanese company specialized in large-scale banking and government systems. They were used to test a proof of concept application called "BookSearch". Testcases were designed for testing functional requirements (most frequent user-scenarios), non-functional (performance: load and time measurements) and regression (re-running the testcases after minor revisions).

vBau. The tools automated the web interface testing in an interoperability scenario for a big German city, focusing on the OSCI-Transport secure protocol and the XBau standard for the building sector. The TTCN-3 test suite acted as a driver for the scenario workflow, and ensured automation by substituting human users.

5.1 Simple Online Banking Application – SOBA

SOBA was developed by Technical University Berlin and Fraunhofer FOKUS Institute. Three different teams realized the requirements [16], implementation, and testing. The testcases were designed to check functional aspects (such as conformance to requirements) as well as non-functional (performance and security).

Requirements Specification. The application enables users to check their bank account balance and to issue transfers to other accounts. The frontend employs a web-based GUI for entry and display of all data. The application is used by multiple parallel users, which identify themselves by an account number and a PIN.

The application has to care for the functionality of three basic tasks: identifying an user; display of account information; performing money transfers. Two Use Cases (UC), "Perform Transfer" and "Display Account Information", are dependent on the third "User Identification".

The identification of an user is done in the UC "Login" by displaying a welcome page and receiving the user's credentials. The application will look up the account and check the PIN. If the credentials do not match the extension UC "Login Failed" will be executed. In this case an informative message is displayed, along with a way to return to the welcome page. The same happens if the account status is "Locked". "Manual Logout" is executed by an explicit user command, whereas "Automatic Logout" will implicitly become operative after a certain time of user idleness. After successful "Login" or "Performed Transfer", the "Display Account Information" UC will always be executed. The user can always perform the latter UCs while logged in.

The "Perform Transfer" UC consists of three Sub-UC: "Enter Transfer Data", "Execute Transfer", and "Transfer Result" After the checks for the transfer data, the user submits the transfer by entering a TAN number. If the TAN is not valid an informative message will appear and the UC "Execute Transfer" will restart; if an invalid TAN is entered three times in a row, the account status is set to "Locked", and the UC "Automatic Logout" is executed. After completion, the UC "Transfer Result" is active and a status message is displayed.

As seen in 4, the application's control flow is quite complex, though the application itself only consists of six screens.

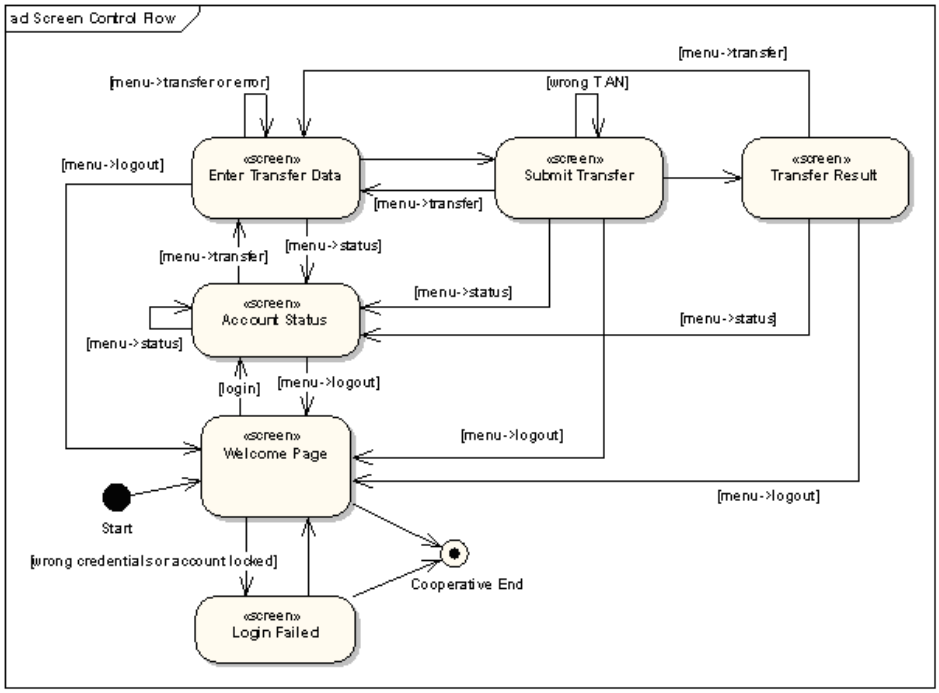

Fig. 4. Activity Diagram - SOBA Control Flow for Application Screens

Implementation of the SUT. A single application server cares for the business logic. The backend consists of a single database, containing the business data. This information encompasses all data about the account, namely: account number, account balance, PIN, account status, owner's full name, list of TAN numbers, counter for consecutive wrong TAN entries.

The client-side test approach consists of two parts: the *TTCN-3 testcases*, which is specified by the tester as TTCN-3 source code, as well as the already defined the *Test Adaptation* layer, acting similarly to a Web browser when communicating with the SuT. We considered three important criteria: conformance to requirements, performance and security.

Conformance or functional testing techniques test the behavior of the system or program. The TTCN-3 type of testing is known as *black-box testing* – data is sent to the system without knowing how the system is implemented, and the system is analyzed according to its output only. The requirements to which the SuT should conform are: user login and logout; after 3 wrong PIN numbers the account will be locked; automatic logout after 3 idle minutes; user cannot log-in twice; correct transitions between the all the UCs; TAN to confirm transfer of money; number of wrong, consecutive TAN entry attempts; transfer is limited to 1000 euro; transfer is only possible between known accounts; correct informational messages.

To test the *performance* of the SUT, we need a mechanism to send data to the SUT at the same time to find out how the SUT will respond, e.g. how many sessions can be run at the same time. Such tests require varying load conditions for the SUT, which can be realized by an ensemble of parallel TTCN-3 test components.

We covered the application *security* by testing how the SUT responds to receiving fake data or a simple access attempt, e.g. to get information about the users without login, to load a webpage without calling the login page, to transfer money with a wrong TAN or without a TAN number.

Implementation of testcases. The tests always need the same initial persistent data (e.g. accounts, PINs) to get meaningful test results. A mechanism to initialize the *SUT* with the same data (every time before test suite execution) is needed. To solve this problem, a TTCN-3 external function was implemented in Java, which executes a SQL script to initialize the database with the same structure and data.

Below we present an example testcase which tests whether a money transfer can be executed correctly. The user must login, go to the "Transfer" page, introduce the target account and the desired amount, enter a unique TAN to validate the transaction and then finish the transfer by checking its result.

```
testcase CTC_Transfer_1_1() runs on WebComponent {
svar HttpRequest req; var HttpResponse response; var Map mp := {}; var XPathMapping xpm;
initDB(); initTTLReq(req, "localhost", 8080);          // initialize database and web-app parameters
openWindow("http://localhost:8080/soba/", req, response, TIMER_DEFAULT); // open first page
checkWindow("/soba/", response);                                          // and login
```

```
putUIValue("account", "11111", req, response, xpm, mp);                 // fill value
putUIValue("pin", "123", req, response, xpm, mp);                       // fill value
click("Login", req, response, mp, TIMER_DEFAULT);                       // perform a click
checkWindow("/soba/LoginControl.jsp", response);                        // navigation check
click("Transfer", req, response, mp, TIMER_DEFAULT);                    // transfer money
checkWindow("/soba/AccountStatusControl.jsp", response); mp := {};      // navigation check
putUIValue("targetAccountNumber", "12345", req, response, xpm, mp);     // fill account number
putUIValue("transferAmount", "50", req, response, xpm, mp);             // give amount to transfer
click("Initiate␣Transfer", req, response, mp, TIMER_DEFAULT);           // perform a click
checkWindow("/soba/TransferControl.jsp", response); mp := {};           // navigation check
putUIValue("TAN","67833", req, response, xpm, mp);                      // enter TAN
click("Submit", req, response, mp, TIMER_DEFAULT);                      // submit transfer request
checkWindow("/soba/TransferControl.jsp", response);                     // check the new page
setverdict(pass); }                                                     // give verdict
```

This high-level view uses convenience TTCN-3 functions implemented in the TWeb module as a Web testing library. The following function is an example:

```
function openWindow(in charstring path, inout HttpRequest req,
  out HttpResponse response, in float timerValue) runs on WebComponent {
 req.uri.path := path; req.method := REQ_GET;
 httpPort.call (load: {req}, timerValue) {
 [] httpPort.getreply (load: {-} value ResponseTemplate) -> value response
   [] any port.getreply { setverdict(fail); stop; }
   [] httpPort.catch (timeout) { setverdict(fail); stop; }}
 if (not (ispresent (response.html1Page))) { log("No␣HTML"); setverdict(fail);stop;} }
```

Test Results. Most of the functions in the SOBA application could be tested successfully. However, a small number did not comply with the conformance tests, i.e. a user can log-in twice, and a misleading page is shown when the transfer is above 1000 euro. The application was tested and sustained an average load of approx. 50 simultaneous users. As mentioned, the security tests were written in order to test whether any information can be obtained without login, or whether a transfer can be made without PIN or TAN – and the web application indeed didn't allow access.

6 Conclusions, Open Issues and Outlook

This article shows how TTCN-3 is an appropriate solution for testing regular Web applications. The practical results show that the design was appropriate and the implementation matched the expectations. The developed software provides black-box distributed reactive testing for HTML and XHTML (including JavaScript) transmitted over the HTTP/HTTPS protocol.

One of the next things to be done is the actual integration of WebTestGUI and TWeb as Eclipse plugins.

Distributed test setups for efficient load, performance, scalability tests are possible using the TTCN-3 test platform. Tools such as TTworkbench Professional enable distribution and creation of parallel user behaviors.

There are some envisioned extensions regarding the Mapping Repository: one repository per application for increased readability and easier management of mappings; avoidance of name clashes for active mappings if they are contained in more Address entries; extended syntax, i.e. multiple URL-like regular expressions, page parameters, current browsing history.

XHTML 2 and XForms, XHTML 5 and WebForms 2.0 are flexible structured XML-based formats designed for the replacement of current HTML/XHTML formats. The advantage of using XPath/XQuery while testing still remains, because these languages are applicable not only to current XHTML, but to any valid XML format. The current architecture doesn't need to be modified, only extended; the Adapter needs to be modified to handle other data structures passing on the HTTP level.

With regard to testing of Web Services, communication of applications that use SOAP protocol can be tested using TTCN-3, because the messages transported are in an XML-based format. Testing Web Services is enabled because JRex support querying Web Services using SOAP through Mozilla Web Services Component[9]. This makes it easier to integrate the testing of Web Services in the current Adapter – the change will be in the format of the document, by switching from HTML/XHTML to SOAP messages/envelopes.

Generally, high-level constructs and quality assessments models can be based upon this framework, which enable measurements of non-functional quality attributes (like response times, capacity, reliability, security), as well as a fine-grained interaction with web users through control of the browser.

References

1. ETSI: TTCN-3 Standard Part 1: ES 201 873-1 V3.1.1 - TTCN-3 Core Language

[9] http://www.mozilla.org/projects/webservices/

2. Grabowski, J., Hogrefe, D., Rethy, G., Schieferdecker, I., Wiles, A., Willcock, C.: An Introduction into the Testing and Test Control Notation (TTCN-3). Computer Networks, Volume 42, Issue 3 (2003) 375–403
3. Xiong, P., Probert, R.L.: A Multi-Approach Testing Methodology in TTCN for Web Applications. (2003)
4. Probert, R.L., Xiong, P., Stepien, B.: Life-Cycle E-commerce Testing with OO-TTCN-3. (2004)
5. Stepien, B., Peyton, L., Xiong, P.: Framework Testing of Web Applications using TTCN-3. International Journal on Software Tools for Technology Transfer (STTT) **10** (2008) 371–381
6. Xiong, P., Probert, R.L., Stepien, B.: An Efficient Formal Testing Approach for Web Service with TTCN-3. In: International Conference on Software, Telecommunications and Computer Networks. (2005)
7. Schieferdecker, I., Stepien, B.: Automated Testing of XML/SOAP Based Web Services. In: 13th Fachkonferenz "Kommunikation in Verteilten Systemen" der Gesellschaft fur Informatik (GI). (2003)
8. Jeaca, D.M.: XML Schema to TTCN-3 Mapping - Importing XML Schema Datatypes into TTCN-3. Diploma paper, UPB Bucharest (March-August 2004)
9. Vega, D., Rentea, C., Din, G.: An Approach of Model-based Testing with TTCN-3: Definitions Generation for Testing of Web Services. In: 4th Workshop on System Testing and Validation. (2006)
10. Schieferdecker, I., Vega, D., Rentea, C.: Import of WSDL Definitions in TTCN-3 Targeting Testing of Web Services. In: 9th International Conference on Integrated Design and Process Technology (IDPT). (2006)
11. Troschutz, S.: Web Service Test Framework with TTCN-3. Master's thesis, University of Gottingen, Institute for Computer Science, Gottingen, Germany, (2007)
12. Schieferdecker, I., Vassiliou-Gioles, T.: Realizing distributed TTCN-3 test systems with TCI. (2003)
13. Schieferdecker, I., Din, G., Apostolidis, D.: Distributed functional and load tests for Web services. International Journal on Software Tools for Technology Transfer (STTT), Volume 7, Issue 4 (2005) 351–360
14. ETSI: TTCN-3 Standard Part 5: ES 201 873-5 V3.1.1 - TTCN-3 Runtime Interface (TRI)
15. ETSI: TTCN-3 Standard Part 6: ES 201 873-6 V3.1.1 - TTCN-3 Control Interface (TCI)
16. Hoefig, E.: Simple Online Banking Application - Requirements Specification. Fraunhofer FOKUS, unpublished script (2005)