Volume 4- Number 2- March 2012 (47-60)

E-business Integrated Test Framework Model

Pasha Vejdan Tamar
Department of Information Technology
Tarbiat Modares University
Tehran, Iran
pvejdan@modares.ac.ir

Abbas Asosheh
Department of Information Technology
Tarbiat Modares University
Tehran, Iran
asosheh@modares.ac.ir

Received: August 10, 2011- Accepted: December 29, 2011

Abstract-- **There is an increasing need for an integrated test framework which can do conformance and interoperability testing in all layers of e-business standards without any dependence on a specific standard. In this paper a knowledge framework was provided for designing a model for e-business integrated testing by combining identified design factors from testing experience and conducting the CEN (*European Committee for Standardization*) GITB (*Global Interoperability Test Bed*) project feasibility study. Also, it was shown that abstracting of test scenarios in a modular manner makes them easily understandable, independent from test beds and standards and, furthermore, more reusable. Embedding a test case tool in the test framework provides the capability of automatic generation of executable test cases, and simultaneously makes them more manageable. A modular test bed design, by considering some interfaces to pluggable adaptors and applying event driven execution model, make it configurable and applicable to the various test types. By embedding management components into the test bed, more controlling and monitoring were provided over the test and its steps as, functional requirements for the current test beds.**

Keywords- E-business Test Framework, Interoperability Test, Conformance Test, Test Case, Test Bed,

I. INTRODUCTION

Before being able to work together using electronic beds such as the Internet, Business partners or associates in a community should know in what information, when, and how to participate. Therefore, defining standards begin by expanding the use of internet. These standards specify the requirements and conditions of proper conduction of electronic interoperability transactions. B2B standards prescribe specifications of business processes, business documents; and the messaging infrastructure. Some of the standards explain specifications for a single layer or parts of functionality of a layer and conversely some others such as ebXML specifications work for all layers. Specification of electronic interoperability standards are mainly grouped in three layers: 1) messaging layer defines specification for packing/unpacking, sending/receiving, reliability, security, speed and quality of exchanged messages, 2) document layer_defines syntax and semantics of the body and header of business documents; and 3) business process layer defines messages' choreography, interactions and flow of roles messages and along with rollback/recovery from the failure of business processes.

Some software solutions which support the development of standards are emerging, but these solutions do not usually cover all the specifications or standards. Also, the cost of replacing legacy software systems is very high for organizations. They want to enable electronic collaboration with others under different standards while maintaining legacy systems, thus they use middleware software. Also, even though software systems are implemented for electronic collaboration under the same standards, in some cases these standards include defects or some ambiguous requirements and in some other cases, they may be flexibly designed for software vendors to be creative and make specializations in some fields. Therefore, a software solution needs to be tested to prevent the emergence of late-time errors, which often leads to complex and costly corrections and unsatisfied users. Testing can find errors; in advance, assess and verify quality of solutions; and also prevent from the problems of collaboration between e-business systems, which may lead to irreversible damage to businesses.

Because of reasons mentioned above, e-business systems are required to be tested from two aspects of conformance to standards and interoperability with each other. Conformance test is testing whether the system can provide all the requirements specified in the specifications or standards. Thus, conformance test items are set exactly according to standard specification and this type of testing can be performed for a single system. Testing requirements for a conformance test are identified and the related test cases are defined in the interaction of domain expert and test engineer. Then, system vendors in interaction with domain and test experts test the designed system for business using the test bed. In addition, the interoperability test tests the ability of a system under certain specifications or standards in the partnership with other systems under the same or different standards, networks and independent from a particular programming language or operating system in order to correctly perform business processes and detect interoperability runtime unknown errors. Interoperability tests should be run between two or more systems to determine whether the systems can perform a general task together in a real operational environment. Interoperability test items should be determined together by the domain and test experts and system vendors with reference to standards and application operational scenarios.

To run tests on electronic collaboration, a framework is required which includes a set of utilities and documents about rules, definitions and documentations for testing design, execution and documentation. Using the concepts of the test, the test cases and test bed development occur. A test case includes the stages and steps which should be performed in a test, the parameters that must be considered at test time and conditions that must be satisfied to show that the system under test is functioning properly. To automate the testing process, test cases are in the form of high-readable machine codes and data which are often based on XML. Test framework includes the required documents about the model, rules and grammar of a test case composition. Composed test cases are used by the test bed to execute and lead test toward testing SUTs. So, the test bed refers to a set of hardware and software tools and components, which are defined by test framework and developed and implemented for a specific test objective in order to test SUTs by test cases.

There are different test frameworks that are designed and implemented to e-business interoperability testing, most of which focus on testing requirements of specific standards or communities, And, even within the same standard, they may address only specific interoperability layer or functions of a layer. The objective of this article was to draw features and designs of a test framework used for integrated test conformance and interoperability of e-business systems in all interoperability layers without affiliation to any specific standards or specifications.

In the following parts, the problems of the test frameworks are represented based on specific e-business standards or communities; then, by analyzing features of the new test frameworks that aim to apply integrity features and refer to the recommendations of the first phase of GITB project, the required design factors of an integrated test framework are obtained. Then, based on those factors, the required concepts are developed for a framework, which included design and production model of test cases and test execution model and the required components of test bed. The discussion is summarized at the end.

II. Review of Existing E-Businesses Test Facilities

Simultaneous with forming community for electronic collaborations and their support, different standards have gradually developed and then testing requirements have raised for the systems developed based on the specification of these standards and communities; thus, testing approaches and frameworks related to each of these specification have also emerged. These initial test frameworks have specific purposes and have been designed for partial or full requirements of a specific standard or community. For example, ebXML test framework IIC1.1 [5,16] which was designed for all collaboration layers of this standard in theory, but has been only developed by organizations such as KorBit and NIST for testing ebXML messaging service specification (ebMS) implementer systems. WS-I test tool is used for assuring functionality and compatibility of the developed web services against the guidance and documents of WS-I profiles. Moreover, Rosetta Net Self Test Kit (RNSTK) is a package provided from Rosetta Net Community that verifies the existing product according to RNIF specification provided from Rosetta Net.

Those test frameworks have problems such as the following cases: they are designed for specific standard testing requirements and are not usable in other standards or specifications or can be used with many challenges. Test cases and test beds strongly depend on each other and are not reusable with other test beds or test cases. They do not cover all testing requirements in all the interoperability layers; instead, they are designed to test the requirements of one or two layers or to test them partially.

On the other hand, there are test frameworks such as TTCN-3[1] from ETSI[2] and TestBATN[3][3,13,14,15] from Turkey which apply modularity and use different adaptors for different standards and specifications and could be used in domains further than their original domains. Also, OASIS TaMIE[4] Technical Committee has represented eTSL[5][4,6] and eTSM[6] [17] since 2006; recently, Xtemp [7][18] defined an event-driven execution model and an XML-based scripting markup for test cases in order to provide appropriate background for fulfilling integrity in the e-business test frameworks. Execution model of Agile Test Framework (ATF) [11] and NIST Athena TestBed are based on eTSL v 0.78.

Finally, since 2007, an international activity has started in European Committee for Standardization (CEN) in order to create a Global Interoperability Test Bed for e-business systems. This activity is supported by most of e-business test correlated organizations such as EIC[8], ETSI, NIST[9], KorBIT, AIAG [10] and IAI. The purpose is to establish a basis for a test framework (i.e., GITB) for effective development of globally distributed e-business test beds. Since October, 2008, the activity has continued in the form of a project with three phases: Feasibility Study, Conceptualization of the target architecture and Realization. The project's first phase ended in December, 2009, and its final document was published in February, 2010, which was entitled 16093 CWA[11]. In January, 2011, the project entered its second phase and its draft CWA was open for public comments until 31 October 2011 [2].

CWA 16093's document included valuable reports from testing requirements in three business domains: 1) "Long Distance Supply Chains in the Automotive Industry", 2) "Health Level (HL7) v3 Scenarios" and 3) "e-Government-Public Procurement – CEN/BII Scenarios" and presented a summary of the existing testing capabilities (testing frameworks and methodologies, testing architecture and test beds), assessment of these testing capabilities with regard to the requirements and the suggested GITB requirements. Also, this document explained alternative approaches for architecting and implementing a global e-business interoperability test bed and provided some recommendations related to e-business testing, e-business standardization, GITB architecture and so on.

It has been expressed in e-business testing recommendations, for future conformance and interoperability testing, there is a need for integrated testing frameworks which do not hard-code a specific standard at any layer (because different communities may use different standards) and are capable of handling testing activities at all layers of the interoperability stack. So, it recommended:

- E-business testing needs comprehensive testing frameworks and methodologies covering test execution and test case model. For this purpose it is necessary: 1) Test case structure and scripting grammar should be standardized. Candidate approaches can be, e.g., the event-driven Test Scripting Model (eTSM) being developed under OASIS TAMIE Technical Committee. 2) Testing frameworks and methodologies need to be further enhanced beyond business document testing in order to cope with requirements related to business process testing in complex scenarios. 3) In developing the testing framework and methodologies start with the requirements and conceptualizations identified in this project to foster shared terminology and common understanding of the needs and key capabilities
- The Global e-Business Interoperability Test Bed should be realized in a decentralized approach as a network of multiple test beds and test services. It recommends that: 1) essential foundations include a standardized testing framework (covering test execution and test case model), a coherent architecture, a common platform and access method, 2) the functional requirements should be implemented as plug-and-play components and leverage existing standards (e.g. transport and communication standards such as ebMS, validation standards such as XML Schematron and query mechanism standards such as XPATH), and 3) emphasis also needs to be given to non-functional capabilities such as modularity and extensibility.

III. Requirements of an e-business integrated test framework

In GITB feasability study general requirements has been estimated and classified for an integrated test bed according to Fig -1.

"Business-level requirements specify the subject of testing (What type of concern to test for?). Engineering-level requirements fall into two categories: functional and non-functional requirements. Functional requirements specify the means by which the testing goal is achieved (How to test?) Non-functional requirements specify the additional concerns under which the testing functionality is or needs to be achieved (e.g., maintainability, modularity, reusability).

[1] Testing and Test Control Notation v3
[2] European Telecommunications Standards Institute
[3] Testing Business, Application, Transport and Network Layers
[4] Testing and Monitoring Internet Exchange
[5] Event-driven Test Scripting Language
[6] event-driven Test Scripting Model
[7] XML Testing and Event-driven Monitoring of Processes
[8] European Interoperability Center
[9] National Institute of Standards and Technology
[10] Automotive Industry Action Group
[11] CEN Workshop Agreement

Finally, the operating environment requirements allow us to relate business requirements to detailed concerns of defining, obtaining, and validating test items within the specific testing environment."[1, 8]

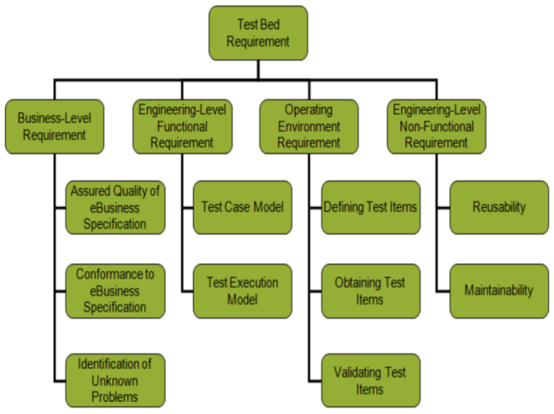
Test Bed Requirement
Business-Level Requirement
Engineering-Level Functional Requirement
Operating Environment Requirement
Engineering-Level Non-Functional Requirement
Assured Quality of eBusiness Specification
Conformance to eBusiness Specification
Identification of Unknown Problems
Test Case Model
Test Execution Model
Defining Test Items
Obtaining Test Items
Validating Test Items
Reusability
Maintainability

Fig 1. Top level test requirements [1]

into some leaf-level requirements. Since the engineering-level requirements directly affect test framewok design, the details of these requirements are given in Table-1. Also, this table shows that, at the engineering level, in order to fulfill each of the functional requirements, which non-functional requirements should be taken into consideration.

These requirements are achived by analayzing two types of eBusiness domains: mature domains and emerging domains. The Health Level (HL7) v3 Scenarios and e-Government-Public Procurement – CEN/BII Scenarios are analyzed as mature domains and Long Distance Supply Chains in the Automotive Industry are analyzed as an emerging one.

IV. *INTEGRATED TEST FRAMEWORK DESIGN*

To fulfill the engineering level requirements of e-businesse testing, a new test framework should be designed. The existing test beds and test methodologies do not fully provide the engineering-level functional capabilities required by the use cases. They lack several non-functional capabilities such as modularity, extensibility and plug & playability. Specifically, there are five cases in Table-1 in which the union of the capabilities of the existing test beds does not satisfy the union of functional requirements of use cases: 1)[Fun-TCE/R01]-1, 2)[Fun-TCE/R01]-3, 3)[Fun-TCE/R04]-3, 4)[Fun-TCE/R05]-2, and 5)[Fun-TCE/R05]-3 [1].

Moreover "The simple addition of missing functional capabilities to existing capabilities may not guarantee interoperability between added components and existing capabilities... As a result, a new test framework should be developed to satisfy the requirements extracted from the three use cases and to enhance the reuse of existing capabilities."[1]

In the testing literature, parts of a test framework typically consist of two general categories of test suite and test bed. A test suite consists of test cases and each test case is a set of steps, variables, and conditions, under which the test user verifies the function of SUT. A test bed is a set of facilities which are used by a user to test SUT(s); also, the test bed does that by processing steps and conditions of the test case. Therefore, the model of test framework is a combination of two test cases and test bed models. “The test case model describes the contents contained in the test case and their structure; the test execution model guides the implementation of a test bed to validate B2B solutions and documents against the regulations and agreements between trading partners."[1]

This resolution is the first step in the test framework modularity and prevents from the hard coupling of test case and test bed while increasing the reusability of test cases. As has been said in the [GITB Requirements2-1] suggestion, ”…a test case for a specific test requirement can be developed independent from a specific test bed system. It makes the test cases more reusable and manageable, i.e., once developed by domain experts, a test case may be reused repeatedly by other test users. In addition, the test case may be used for other test bed systems if the systems conform to the GITB technical requirements.”

Furthermore, designing test cases is prior to test execution model because “most functional requirements for test execution depend on the functional requirements of the test case design. For example, the definition of “capability of test preparation and setup” for test execution depends on how “test configuration information” is represented. Therefore, the structure and grammar necessary to represent a test case should be considered first."[1]

Therefore we continue first by design model for test case and then we will explain the execution model for an integrated test framework.

TABLE 1. Engineering level requierments [1]

Key Capability Index for Engineering Level Functional Requirements		Reusability		Maintainability	
		Modularity	Plug & Playability	Extensibility	Robustness
Test Execution Model	**[Fun-TCE/R01] Capability of test preparation and setup**	√			
	1) Capability of providing the setup information to SUT(s)	√		√	
	2)Capability of requesting SUT's parameters and information	√		√	
	3)Capability of test case customization	√			
	4)Capability of configuration of setup information	√		√	
	[Fun-TCE/R02] Capability of controlling test steps	√			
	1)Capability of display of test flow and test progress	√			
	2)Capability of requesting/storing user's information	√			
	3)Capability of binding user's information into test case	√			
	4)Capability of manual execution of test steps	√		√	
	[Fun-TCE/R03] Capability of message exchange		√		
	1) Capability of sending/receiving message payloads?		√		√
	2) Capability of uploading/downloading message payloads		√		
	3)Capability of capturing message		√		
	[Fun-TCE/R04] Capability of message pre/post-processing	√			
	1)Capability of decomposing message	√		√	
	2)Capability of retrieving the value from message	√			√
	3) Capability of generation message template from schema	√		√	
	4) Capability of generation test data for a specific message template	√		√	
	5) Capability of message transformation	√		√	
	[Fun-TCE/R05] Capability of message pre/post-processing		√		√
	1) Capability of detecting unknown problems		√		√
	2) Capability of employing the existing validation engines		√		√
	3) Capability of recovery from errors		√		√
	[Fun-TCE/R06] Capability of reporting		√		
	1) Capability of display of error location		√		
	2) Capability of display of test log information		√		
	3) Capability of display of the detail test result		√		
	[Fun-TCE/R07] Capability of B2B system emulation (optional)		√		
	1) Capability of emulation of an arbitrary unit		√		
Test Case Design	**[Fun-TCM/R01] Capability of representing test configuration information**	√			
	1) Capability of representing declaration of messaging protocol to	√		√	
	[Fun-TCM/R02] Capability of representing test procedural information	√			
	1) Capability of representing message to be sent	√		√	
	2) Capability of representing message choreography	√		√	
	3) Capability of representing conditional expressions (test step) for test case	√			
	4) Capability of representing iterative expression (test step) for test case	√			
	5) Capability of representing manual steps	√			
	[Fun-TCM/R03] Capability of representing test verification information	√			
	1) Capability of using external documents for verification(e.g. XML Schema)	√		√	
	[Fun-TCM/R04] Capability of representing test suite which contains a set of test cases				
	1) Capability of representing precedence relationships between test			√	
	[Fun-TCM/R05] Capability of representing test data				
	1) Capability of representing of user's defined values			√	
	2) Capability of representation of automatically generated values (i.e. using metadata)			√	

A. Test Case Model

To achieve integrity in the e-business testing, the existing problems reported from design of test cases in the initial test tools and frameworks must be somehow addressed and resolved. Major issues include:

- *The gap of understanding:* since the specifications of standards are written in the form of an informal language rather than a formal one, it leads to a gap in understanding and the incidence of misunderstanding between the test agents (test users, software vendor and test engineers) in perception of concepts of standards.
- *Test dependency:* the existing test suites have been designed for a specific testing tool or for a specific standard specification and they may not be reused for other test tools or other standards.
- *Hard coupled test information:* a test case at runtime includes a lot of information about test procedure, assertion and environment. In the existing test suites, this information is hardly coupled with each other. However, these test cases work well in test tools, but their understanding for test users and also their maintenance are challenging.
- *Fixed coded test cases*: once created for a test bed, initial generation of test cases can be used repeatedly. But, any change in them such as customization or data changing during the test is impossible; for making changes, test cases should be re-reviewed and re-coded, so does the test.
- *A Long and costly production process*: production and maintenance of test cases are the most expensive and time-consuming part in the e-business testing since there is a need to analyze the behavioral, technical and content aspects of standards; also, knowledge about the test such as features of test beds, execution model and scripting language is required and, finally, long scripts should be produced for a lot of tests.

The set of above issues makes the process of producing test cases complex, lengthy and costly, difficult to maintain, less reusable, without customization and applying the test runtime data.

More, it will be shown that, application of design factors such as abstracting, modularity, event-driven, automation and capability of customization in an integrated test framework design can resolve the existing problems of test suites.

1) Test Case Structure (Layers and modules)

To resolve the understanding gap and dependency problems in the test cases, layering can be reused. Layering in the test case production is done by separating it into two "abstract test case" and "execution test case" layers. Abstract test suite is for creating common understanding among test human agents (domain expert, test user and test engineer). The form and syntax of abstract test suite may vary according to users or specifications. Abstract test suite is made from base of standard and application specification in interaction with the domain expert and test user. Thus, abstract test suite can be read and understood by the domain expert and test user. Therefore, it is made of compositions and terms that are familiar to both of them. An "abstract test suite" in a test framework based on test environment information (including partners, services and related messages) may be in interaction with a test case generation tool and may be converted to the readable, interpretable and executable scripts by test bed called "execution test suite".

In the [GITB Requirements 2-3] suggestion, the advantages of this type layering are expressed as follows: “1) an abstract test case can be developed in a more generic manner without considering details of test bed system and 2) many test cases used for the existing test frameworks may be executed by a GITB test bed as an executable test case.” Also, abstracting makes automation and management of test case production process easier using a tool. Using its ability, the capabilities of test case customization, making specific messages templates and using test users' data are also provided.

Test case separation in two abstract and execution layers is an issue that is currently used by TestBATN and ATF test frameworks. Abstract layer in the TestBATN framework is called "test scenario".

As said earlier, a test case includes information about procedures, assertions and environment of test. Hard coupling of these groups of information in the test suites makes them too big for maintenance and too hard for reuse. By breaking down each type of this information into separate modules, these problems can be overcome. Procedure module in the abstract test case “…includes the partners’ life cycles and actions during testing. Actions are abstract descriptions and contain no message instances. For example, the usage script may say “A buyer sends a purchase order message to a supplier.” The specific buyer, purchase order, and supplier instances are not yet specified. On the other hand, the procedural script in the executable test case represents a business transaction that will be executed and contains specific instances and references to the actors in the business process."[10].

Before verifying a test item such as a document or message against an assertion, some conditions must be provided. “These activation conditions render the verification rule independent from the testing procedure since the rule is not activated at a specific step of testing procedure. Consequently, verification scripts may be reused readily within a new testing procedure because the verification script is independently executed by the events during the test procedure."[10]. Assertion module in the abstract test case is human readable and its verifier may be unknown. When that verifier is known, and assertion codes are developed by an executable language, it will be verification script of the executable test case.

Test environment information introduces and identifies test participants and the services that each participant provides or use along with the specification of type and format of messages used by each service. Test environment information in the abstract test case does not have any information of specific participant; instead, it specifies them as "seller application" or "buyer application" and so on. Also, templates of messages are defined without creating specific instances. But, runtime information includes test harness based on specific participant instance information, specific messages' instance created based on message templates and test users' information.

Designing test cases in two abstract and executable layers is indeed done by applying [GITB Requirements 2-3] suggestion. In order to make them modular in the above manner, [GITB Requirements 2-4] suggestion of CWA 16093 is applied. Thereby, [Fun-TCM/R01], [Fun-TCM/R02], [Fun-TCM/R03] and [Fun-TCM/R05] requirements of those documents are also addressed; accordingly, some of test execution requirements are affected.

2) *Executable Test case structure and scripting grammar*

The execution test cases are conversion of abstract test cases so that they can be executed by the test bed. Therefore, it is necessary to standardize their structure and syntax because only the standardized test cases can be identified, interpreted and executed by means of a test bed; i.e., the underlying common understanding between test agents. Also, it provides the possibility of test repeating and the portability from a test bed to others. Hence, a test suite is generally a series of data and codes written in XML scripts according to [GITB Requirements 2-2].

CEN 16093 CWA Report suggests the event-driven scripting model (eTSM) developed by OASIS TAMIE TC as a candidate approach to standardization. The recent work of this committee has been provided with the title of Xtemp, which is briefly reviewed below:

Xtemp

Xtemp has a simple structure and a limited but sufficient number of instructions; so, its implementation is easy for extension and evaluation. It is based on XML that leads scripting to work normally with most of e-businesses profiles, which are XML-based. Also, portability of scripts, models and engines of test is guaranteed. Its execution model is event-driven and independent from time, which makes test scripts equally applicable to validate both real-time and deferred events. Coordination of test case execution into the test suite and test workflow status has been shown as events. This makes management of the large test suite much easier. It is independent from any platform and protocol. It is also provided and supported by OASIS TaMIE and its documents are available.

Using Xtemp structure and grammar enhances the satisfaction of engineering requirements stated in the abstract layer, specially [Fun-TCM/R02 -1,2,3,4,5] and [Fun-TCM/R02 -1].

3) *Test Case Generation and Management tool*

As said earlier, application of abstracting and consequently modularity to the structures of test cases provides a suitable background for their automatic generation and effective management by a test case generation and management tool. Using Test Case management tool has been carried out in earlier TestBATN test framework as Test Design GUI, which is used for dynamic definition of test scenarios for creating corresponding TDL[12] [13, 14, 15] Furthermore, NIST has implemented a test case generation method using a tool in the healthcare domain, the essential goal of which is to facilitate specification, generation and traceability of test cases [10].

Using a tool in generation and management of test cases leads to the automation of test case generation process and makes it possible to develop, implement and maintain test cases and auxiliary materials of different standards and group them in test suites. Also, the capability of test case customization and application of user actual data along with creating message instances from message templates become feasible.

As can be seen in Fig -2, the working method in a test case generation and management tool can be this way. By using the designed abstracting forms' instance such as an abstract test rule form for representing business process, message template form, environmental information template form and test abstract assertion form in the abstract test case's specific GUI, requirements of test cases are defined and saved in the abstract modules' corresponding tables in the abstract test cases' repository. These abstract test cases which have been created for a standard or a type of test could be reused for generating executable test cases for different test models and test conditions.

Furthermore, test agents in interaction with a management GUI of the tool can do some extra and management actions as follows: in test configuration and runtime: getting additional test configuration data, creating message instance from message templates, grouping test case in test suites, test case customizing, applying test harness configuration data, determining test pluggable modules such as verification or messaging modules, applying users' proprietary data and creating validation context files with the expected values. Then, the tool using abstract test cases and applying this extra information based on a scripting standard such as XTemp and its constructs can generate execution test cases.

[12] Test Description Language

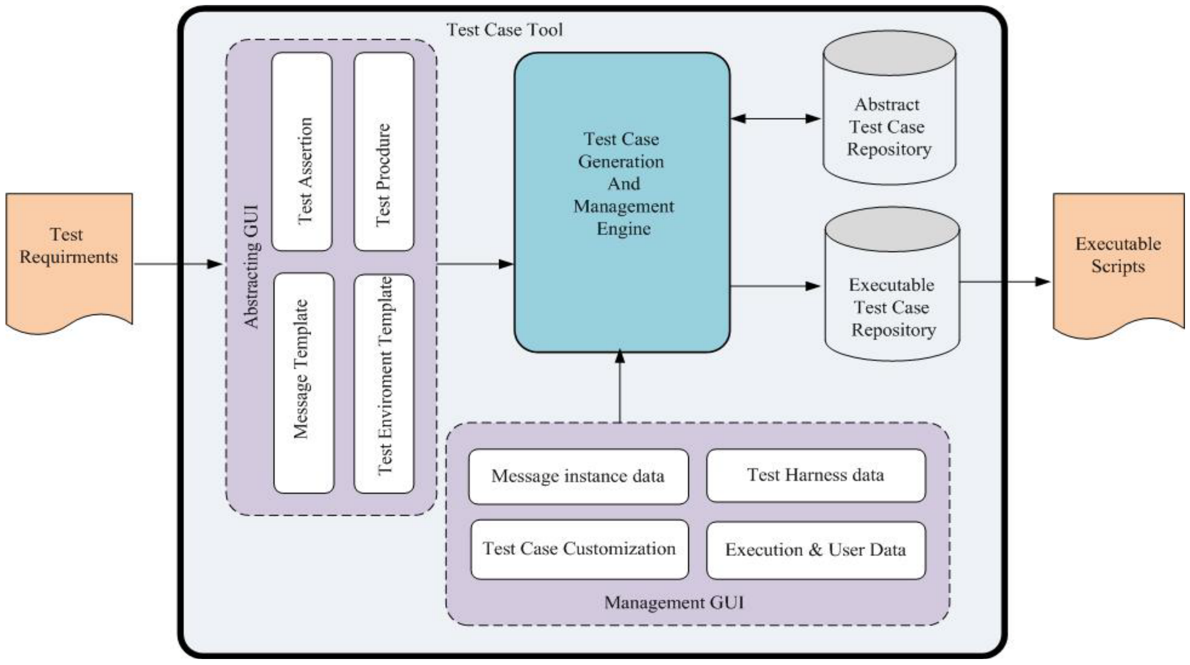

Fig 2. Test Case Tool

Test case tool provides the ground for controlling and changing test steps which helps to test control and running in more or less than its prospected steps. Test case tool can be used to fulfill all GITB engineering functional requirements, specially [Fun-TCM/R04] and [Fun-TCM/R05]; also, it helps to fulfill non-functional requirements' modularity and extensibility in test cases.

B. Execution Model in Integrated Test FrameWork

Execution model is architecture of testing tools for test execution. This model shows the test bed components and their orchestration together in interaction with the test environment to fulfill test process. The integrated test bed execution model wants to show that it is able to test any system under any standard and in any interoperability layer and that it can enrich the testing process by providing test management capabilities.

Integrated test bed architecture should be able to cover a range of standards and all interoperability layers and should be implemented quickly. This requires the existence of *pluggable* and *Plug-n-Play* test components. Pluggable components provide the capability of testing different standards; also, they cause some components of the test to be reused commonly for different standards, solutions, test modes and test configurations. A precise and specified definition of test component interfaces makes test bed capable of automatically detecting and connecting the test components which are adaptors to the corresponding interfaces for different standards and specifications. Plug-n-Play feature makes it easy to setup test bed and automation of test process.

To be used in different test environments, the test bed structure should be lightweight and easily maintainable. Design factors such as *modular design* and *event-driven* execution model can be of help in this process. Modular design provides better resource management and reduces maintenance cost while increasing reusability. Also, an event-driven structure causes a test bed with agility to manage different resources because components can interact with each other without any direct connection between themselves. It also manages a range of components without any heavy load over the test engine. "When a component attempts to interact with another component, it sends data to an *event board* instead of the target component. The *event board* stores various types of interaction data as events so that every component can inquire and retrieve a specific event. All the activities of the *pluggable components* and the *test infrastructure* are coordinated via events."[9]

An efficient test bed must have appropriate interaction with test users on setting up, running and completing test time in order to be able to obtain or provide the required information from them and provide the necessary assistance for error correction in their application systems. The existence of *management* and *detailed test report* components can be useful in these cases. Test management component provides the capability of controlling test steps by the user, setting up test bed, applying actual and not predetermined runtime data, supplementary utilities such as graphically displaying the test flow, test evaluation by users and test bed interactions with the SUTs. Furthermore, the existence of detailed test reporting component causes additional reporting of test log and result and can get detailed reports from the exact locations of errors, detail of test results and analytical results.

According to the above description, an integrated test framework's conceptual architecture model could be seen in Fig-3. In this design, some experiences were applied in e-businesses test. Like IIC1.1 framework, the central component of the test is test engine; however, messaging and evaluation components like TestBATN and ATF frameworks could be considered as pluggable adaptors. It can be

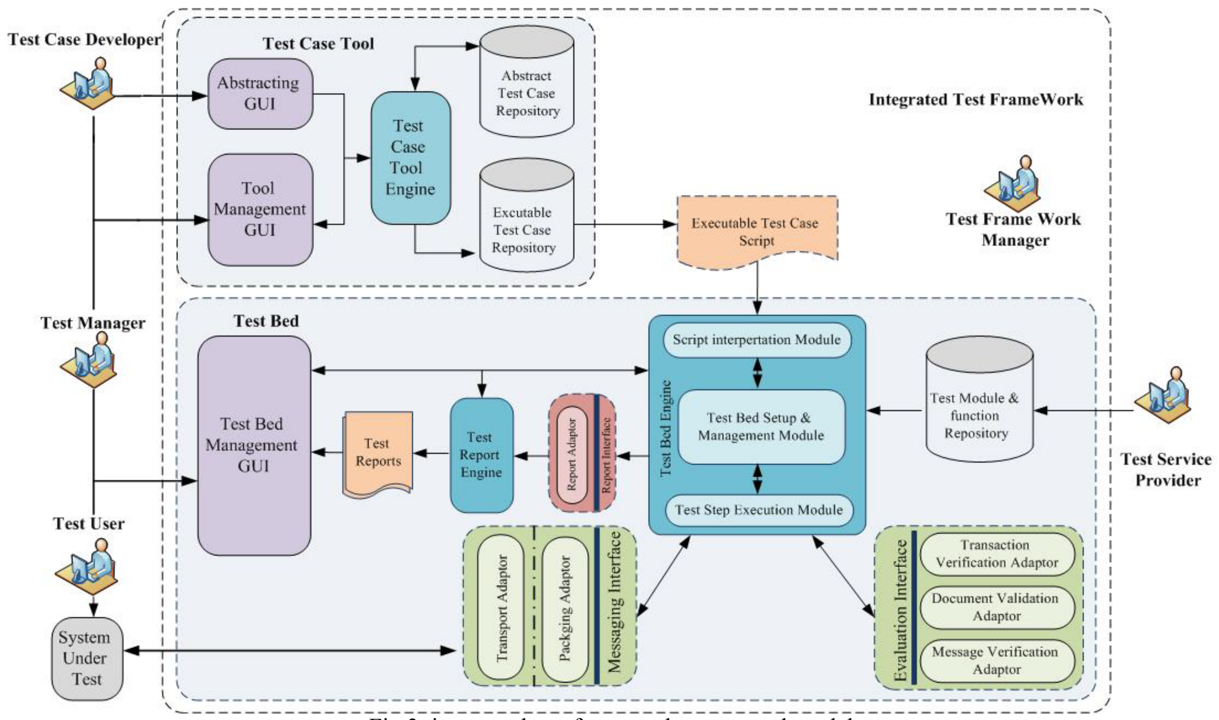

Fig 3. integrated test framework conceptual model

also considered that interfaces were well-defined and adaptors must be properly bound to them so as to be detected and connected by the test bed in the Plug-n-play manner. By selecting Xtemp execution model for executive test cases, every event in the bed and bed interactions with the outside world were necessarily managed by an event board and test engine. Hereby, features such as reusability and easy management of various resources were obtained. Also, like TsetBATN framework, by designing test management GUI component, test run-time control is provided. By embedding report generator engine and defining its functional specifications, lack of available beds in the test reporting is addressed. By categorizing test verification and validation adaptors, test coverage is increased.

Integrated test bed components included *central component*, *test adaptors interfaces* and *test management components*. Central component is the module pre-defined by the execution model. Test interface adaptors are discoverable modules. Managerial components are for controlling, conducting and providing the test's required outputs.

1) *Test bed central component*

Test engine is the test central component. Here, it was considered to have the test engine structure similar to ATF; however, its managerial duties are extended. Test engine is the test bed brain that controls the entire test process. It reads and interprets a test case, coordinates interfaces and other test components, conducts the execution of test with the SUTs which should be tested and also interacted with management components and provides test management. The test engine has the following three major internal modules:

- *TEST SCRIPT INTERPERTATION MODULE:* module reads the presented executable test case and then separates it to test procedures to define test execution steps, test assertions to evaluate the messages, documents and transactions and configuration information to set up a test-bed and discovers and plugs appropriate test interface adaptors, and finally, to put them to the Test Bed Setup and Management Module.
- *Test Bed Setup and Management Module:* This module allows the executable test case to the interpreter module and uses its configuration profiles for discovering and connecting the appropriate test adaptors. It defines a process model for the test according to test procedure, discovers instances of pluggable adaptors, management components, test engine, their messages and calling sequence. Then, puts it to Test Steps Execution Module. Also, this module in interaction with test users and manager using Test Bed Management GUI provides control of test steps and test effective management.
- *Test Steps Execution Module:* this module interprets the Test Bed Setup and Management Module defined process and executes each activity of process by calling a function.

2) *Interfaces of Test Adaptors*

Interface of test adaptors provide test bed interactions with the SUT and test pluggable module, test engine places and call them dynamically according to configuration specifications. In the integrated execution model, they are reference models specific to messaging protocols or testing assertion phrases. A test bed could have multiple adaptor instances to each interface and recall appropriate instance during test execution inside test service at each step. By designing these interfaces, [GITB Requirements 3-1] from CWA16093 report was used which led to the fulfilment of [Fun-TCE/R03] to [Fun- TCE/R05] engineering-level requirements.

The following interfaces are defined for an Integrated Test Framework:

- *Messaging Adaptors' Interfaces*

  TestBATN test framework [15] has broken interface functionality of messaging adaptors to two interfaces which makes their corresponding adaptors more reusable and lightweight.
  - *Transport adaptors' interface*: this interface facilitates using adaptors for receiving or sending messages by protocols such as TCP, HTTP, SMTP, etc.
  - *Packing/Unpacking adaptors' interface*: this interface makes it possible to use adaptors which are used in packing and unpacking messages according to higher layer communication protocols such as SOAP or ebMS. With this mechanism, to support an SOAP connection over HTTP, one HTTP adaptor is selected to transport interface and one SOAP adaptor is selected for packing interface. From this adaptor's descriptive files, test engine knows that HTTP adaptor divides it into two pieces after receiving messages. It is also clear that SOAP adaptor takes these inputs and produces four message pieces in which the content of the first piece is HTTP header, the second is SOAP header, the third is SOAP body and the last includes appendices. For this output pieces, adaptor descriptors also describe data types. In this way, evaluation of a message contents is easier to a message evaluator adaptor.
- *Verification of Adaptors' Interfaces*

  This interface is designed for plugging test verification and evaluating adaptors to test bed and general classification which can be as follows. This type of classification is used to show verification coverage of an integrated test bed; many of them could be plugged and used simultaneously by a test bed.
  - *Content validation adaptor*: these type pluggable adaptors are used in the validation of contents against a schema in order to generate a verdict and a structured test report about validation; e.g., XML schema validation adaptors or Schematron validator.
  - *Message verification adaptor*: these types of pluggable adaptors are used to perform complex testing over any content of messages. And, its example is XPATH verification adaptors.
  - *Transaction verification adaptor*: this type of adaptor is designed to verify transactions of a process. Currently, no well-known verification has been introduced for this type of adaptor; this is one of the main weaknesses of the existing test beds, which is emphasised in CWA 16093 report.

3) Test Bed Management Components

Management components have been embedded in test bed to enrich the control and management of test execution by the test manager and help test users in preparing various reports for the identification of faults in their systems.

- *Test bed management GUI*

  The idea of this component is driven from TestBATN framework [14] , but more functionality has been defined to cover testing requirements. This component is an interface between Test Bed Setup and Management Module and Test Report Engine with test managers and test users for supervising test execution, controlling test routine, managing test bed resources and the required test reports. It is intended for the following functional characteristics:
  - Updating test bed materials such as deploying libraries of adaptors and new functions,
  - Enabling users to view and lead the execution of test cases,
  - Controlling the SUTs according to the entered instructions during the testing by test bed manager,
  - Interfacing between test users and Test Report Engine to get test verdict and reports,
  - Controlling test steps to run the test step by step or even stop it,
  - Providing graphical and displaying facilities such as sequencing model and test flow to achieve real-time monitoring to test executions.
- *Test Report Engine and its Interface*

  Report engine is used for reporting test summary, test results and other detailed and analytical reports for test users. Report engine uses event-board test transactions to produce necessary test reports. This engine also has interface for several report adaptors which support different report purposes according to test use cases. Test Report engine allows this output for test users through GUI. Some of the expected functional features from these test reporting components are:
  - Concluding and providing test final verdict,
  - Providing verifications of test steps,
  - Showing accurate error locations,
  - Providing Test Log report, and
  - Providing analytical reports and business intelligence.

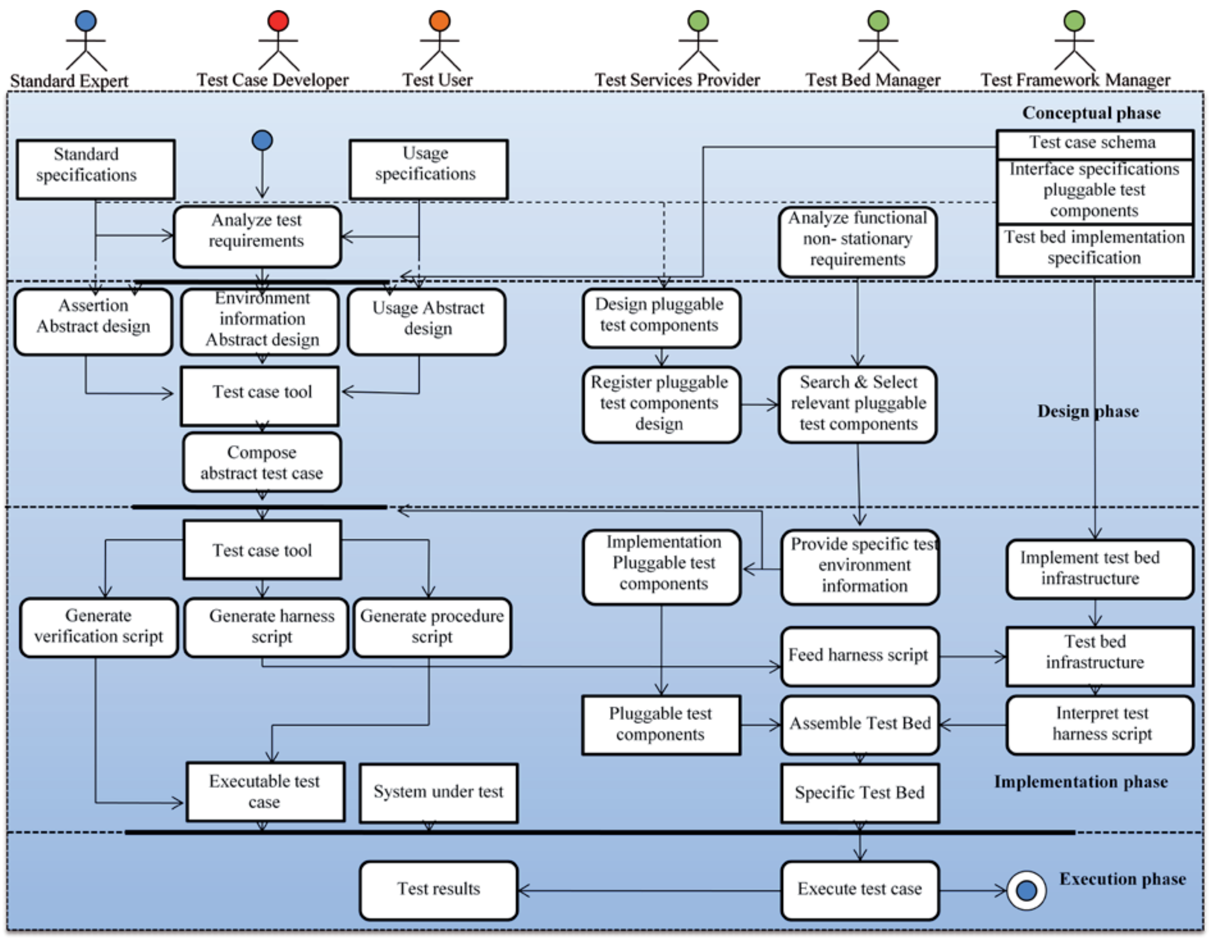
Standard Expert
Test Case Developer
Test User
Test Services Provider
Test Bed Manager
Test Framework Manager
Conceptual phase
Test case schema
Interface specifications pluggable test components
Test bed implementation specification
Standard specifications
Usage specifications
Analyze test requirements
Analyze functional non- stationary requirements
Assertion Abstract design
Environment information Abstract design
Usage Abstract design
Design pluggable test components
Test case tool
Register pluggable test components design
Search & Select relevant pluggable test components
Design phase
Compose abstract test case
Test case tool
Implementation Pluggable test components
Provide specific test environment information
Implement test bed infrastructure
Generate verification script
Generate harness script
Generate procedure script
Feed harness script
Test bed infrastructure
Pluggable test components
Assemble Test Bed
Interpret test harness script
Executable test case
System under test
Specific Test Bed
Implementation phase
Test results
Execute test case
Execution phase

Fig 4. Integrated test frameworks activity diagram

Test Engine, Test Report Engine and Test Case Tool Engine components are neither standard-specific nor use-case-specific; so, they are designed as infrastructure components and are according to CWA 16093 [GITB Requirements 3-3]. Also, these components Along with Test Bed Management GUI and Test Case Management GUI provide fulfillment capability of [Fun-TCE/R01] for [Fun-TCE/R06] requirements. Since Test Report Interface component can vary from test types and use cases, its design is also pluggable and corresponds to [GITB Requirements 3-2].

V. Testing Process in an Integrated Test Framework

Fig-4 shows activity diagram of the integrated test framework architecture with phases and supported processes. It is the same as the one defined for ATF [9, 12], with the simple deference of adding test case tool role. There is a design phase between the conceptual phase and implementation phase, the activities of which are independent from implementing a special test bed; therefore, this phase artifacts are reuseable in different tests. Also, there are three roles representing test bed agents which are: Test Services Provider, Test Bed Manager and Test Framework Manager. These roles and their activities lead to more modularity and reusability.

In the following sections, some of the supported key processes are presented by applying them to the test process of "Sync Shipment Schedule" messaging in AIAG IV&I[13] as a simple expample:

- *Analyzing Test Requirements*: by referring to the standard specifications, Test Case Developer specifies the functions of SUT which are required to be verified. In the present example, "Sync Shipment Schedule" message was issued when the customer made a decision that the supplier should replenish the consumed kanban(s). This message should be conformed to OAGIS [14] -AIAG BOD (SyncShipmentSchedule.xsd in OAGIS9) and messaging between the parties should be conformed to the RAMP [15] profile. Fig-5 shows the interaction diagram in testing customer application. So, two test requirements were considered: conformance test to BOD sent by customer application against OAGIS-AIAG BOD schema and messaging behavior in conformance with RAMP profile [7].

[13] Inventory Visibility and Interoperability
[14] Open Applications Group Integration Specification
[15] Reliable Asynchronous Messaging Profile

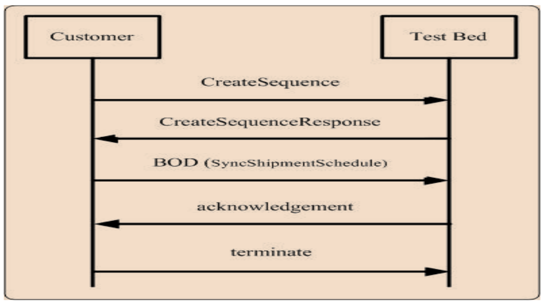

Fig 5. RAMP interaction diagram

- *Abstract Test Case Composition:* Test case abstracting includes abstracting test procedure and test assertions and defining templates of test environment information and test messages. Test procedure abstracts are explained in pseudo codes format in Table-2; afterwards, they can be mapped to executable scripts.

TABLE 2. Abstract Test Procedure

StandardId=1
ProcdureId=1
ProcdureDesc="Customer-SyncShipmentSchedule"

Partner(A," Customer");
…………………
Start(A);
Wait(2s);
read("CreatSequence");
Verify(" CreatSequence");
send(A," CreatSequenceResponse");
Wait(2s)
read("syncShipmentSchedule");
Verify("syncShipmentSchedule");
send(A," Acknowledgment");
wait(2s)
read(Terminate);
Verify("Terminate");
ReportResult();

Test assertions are taken from standard requirements. For example, Test Assertion for conformance test to BOD is demonstrated in Table-3.

TABLE 3. Test Assertion abstract

TestSuiteId =1 *AssertionId*=1 *AssertionDesc*=”BOD Conformance ”
Expected event: receiving BOD message instance from SUT Comparing condition: compare it with reference instance

Moreover, environment information pattern for this test is as given in Table-4.

TABLE 4. Test Environment pattern

TestSuiteId =1 *EnviromentId*=1 *EnviromentDesc*=”Customer testing environment ”			
Party	Message	I/O	protocol
Customer App	CreatSequence	O	RAMP
	CreatSequenceResponse	I	
	syncShipmentSchedule	O	
	Acknowledgment	I	
	Terminate	O	

Test Case Provider produces an abstract test case in interaction with Test User and Standard Expert and by using Test Case Tool; then, they are saved in abstract test case repository.

- *Design and Register Pluggable Test Components:* In this testing, the pluggable adaptors are required as below: A BOD comparison with the sync Shipment Schedule document validation, a SOAP messaging adaptor with RAMP messaging, a HTTP adaptor with message transportation and also a XPATH engine with RAMP profile verification over the received messages. Test Service Provider designs these adaptors in accordance with specification of test interfaces and, then, deploys and registers them in test framework to be accessible. Table-5 shows a sample of adaptors' registration information.

TABLE 5. Pluggable Addaptor registry

Interface	Id	Name	description	address	..
Message Verifier	1	XPATH Validator	The interface definition of the XPATH verifier	http://localhost/testbed/verifier/xpath	..
Messag Transport	2	HTTP	send/Recives messages in HTTP format	http://localhost/testbed/Messaging/HTTP-1	.
..	.	..	..	..	.

Now, after designing and registering the above adaptors, Test Bed Manager searches and selects their specification to be used in the next step.

- *Generate Test Harness and Deploy Test Bed:* After designing test cases abstract and test pluggable components, Test Bed Manager provides test environment data which include: configuration information data describing the participants and destination SUTS (Port, IP, URL), pluggable test components, the protocols and the schemas used by business documents and other necessary information to test bed mounting. Then, these data are fed to the Test Case Tool and test harness script is generated by it. After that, this script is fed to test engine and it configures test bed automatically by calling and plugging test components.

Also, in this stage, Test Case Developer generates messages which should be sent by test bed such as CreateSequenceResponse from their templates existing in the test case tool abstract repository.

- *Generating Executable Test Cases:* Executable test cases include test verification and procedure scripts. Test Case Developer generates them by a test case tool; also, the test case tool does that by mapping abstract test cases to a scripting language constructs such as XTemp.

- *Executing Test and Test Reports*: Finally, when all components are mounted and test parties are ready, test engine starts test execution by calling functions in correspondence with test script steps. Test report engine provides test reports (verdict and detailed reports) from testing stages.

VI. Conclusion

In this article, it was mentioned that, in e-business testing, there is a need for an integrated test framework currently in order to provide the capability of conformance and interoperability testing in all layers of e-businesses wholesale without any dependence on any standards or specifications. Then, the current testing experiences were reviewed and summarized and requirements and suggestions of CEN CWA 16093 document's engineering level were made as an official reference. Finally, the article proceeded to provide a conceptual model for an integrated e-business test framework.

The first step was to model test cases using abstracting and modularity factors. Test cases were considered in two layers; the first one was the abstract layer and the second one was the executable layer. The abstract layer was for abstracting test requirements to make common understanding among test agents; it also provided a suitable context for the generation and management of test cases by a tool. To enhance reusability, test cases were separated into three modules of test procedure, test assertion and test environment. By providing a test case tool, it was shown that, in addition to providing abstracting facilities, this machine along with test agents in an intermediate step could accept test specific and environment information by abstract test cases from its repository and could map them to executable test cases. Accordingly, it speeds up and facilitates generation process of test cases and provides capabilities such as: customization of test cases and applies test users' data for test cases.

The modularity and event-driven factors were applied to the test execution model. The components and interfaces of test execution model were grouped in three categories of Test Central Component (Test Engine), Test Adaptors Interfaces (Messaging Interfaces and Verification Interfaces) and Managerial Components (Test Bed Management GUI, Test Report Engine and its interface). This modular and component oriented structure made test bed adaptable for different test requirements. Test Engine was the test bed infrastructure and was independent from test type. It had three major roles of a) interpreting test cases, b) configuring test bed and c) managing test execution. Test interfaces defined high level functionalities and requirements about the corresponding adaptor instances, which made it possible to implement an interface in different ways while implying test bed for testing different standards or specifications and test requirements in e-business different layers. Moreover, a Test Bed Management GUI in interaction with Test Engine made it feasible to have more control and monitoring over test steps and process. Test Report Engine and Test Report Interface provide detailed and analytical reporting capability for different test usage.

Acknowledgment

The authors would like to thank ICT Research Institute for providing their e-commerce lab, which was founded in 2010 at Tarbiat Modares University.

[1] CEN WORKSHOP AGREEMENT CWA 16093, 2010, " *Feasibility Study for a Global eBusiness Interoperability Test Bed(GITB)",* ftp://ftp.cen.eu/CEN/Sectors/TCandWorkshops /Workshops/CWA16093TestBed.pdf

[2] CEN WS/GITB doc N006, 2011, " *Terms of Reference for the GITB–Phase 2 Project Team*", http://www.ebusiness-testbed. eu/dynamics/modules/SFIL0100/view.php?fil_Id=1010

[3] DOGAC ASUMAN et al,2010," *Electronic Health Record Interoperability as Realized in Turkey's National Health Information System* ", Methods of Information in Medicine (2011) Volume: 50, Issue: 2, Pages: 140-149

[4] Durand, J. (2007) "OASIS ebXML IIC TC." *Event-driven Test Scripting Language. http://kavi.oasis-open.org/committees/download.php/22445/eTSL-draft-085.pdf.*

[5] Durand Jacques & Michael Kass, *The ebXML Test Framework And the Challenges of B2B Testing*, http://ebxmltesting.nist.gov/

[6] Durand J., Kulvatunyou S., Woo J., & Martin Monica J., 2007," *Testing and Monitoring E-Business using the Eventdriven Test Scripting Language*", www.mel.nist.gov/msidlibrary/doc/eTSL.p df

[7] FERRIS Chris & FLOWER Tim, 2005, " *Reliable Asynchronous Messaging Profile Version 1.0*", http://www.ibm.com/developerworks/webservice s/library/specification/ws-b2b/ IBM-WS-RAMP-20050826.pdf

[8] IVEZIC Nenad, et al, 2009," *Towards a Global Interoperability Test Bed for eBusiness Systems* ", Proceedings of the 2009 eChallenges Conference Istanbul

[9] IVEZIC Nenad, WOO Jungyub & CHO Hyunbo,2010, " *Towards Test Framework for Efficient and Reusable Global e-Business Test Beds*", In Proceedings of I-ESA 2010 Conference, Coventry, UK, 2010.

[10] IVEZIC Nenad & WOO Jungyub,2010, " *Testing Interoperability Standards – A Test Case Generation Methodology*", proceedings of the international conference on interoperability for enterprise software and applications A-ESA 2010

[11] JUNGYUB Woo , June 2007,*Agile Test Methodology for B2C/B2B Interoperability* ,Department of Industrial and Management Engineering Pohang University of Science & Technology

[12] JUNGYUB Woo , Nenad Ivezic & Hyunbo Cho, 2011, "*Agile test framework for business-to-business interoperability*", Springer Science+Business Media, Inf Syst Front, DOI 10.1007/s10796-011-9303-3

[13] NAMLI Tuncay, el at,2008," *Testing the Conformance and Interoperability of NHIS to Turkey's HL7 Profile* ", 9th International HL7 Interoperability Conference (IHIC) 2008, Crete, Greece, October, 2008, pp. 63-68.

[14] NAMLI Tuncay, ALUC Gunes & DOGAC Asuman,*2009," An Interoperability Test Framework for HL7-Based Systems "*, IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, VOL. 13, NO. 3

[15] NAMLI Tuncay, DOGAC Asuman, SINACI Ali Anil & ALUC Gunes,2009, " *Testing the Interoperability and Conformance of UBL/NES based Applications* " , Middle East Technical University

[16] OASIS ebXML Implementation, Interoperability and Conformance Technical Committee(IIC), 07 March 2004,*ebXML Test Framework Committee Specification*, Version 1.1, OASIS http://www.oasis-open.org/committees /*download.php/9888/IIC_ebXML TestFramework_v1.1_10_11_04_final.pdf*

[17] OASIS TAMIE, 2010, "eTSM-wd-rev06", http://www . oasis-open.org/committees/download.php /37387/ eTSM-wd-rev07.pdf

[18] OASIS TAMIE, 2011," *XTemp: XML Testing and Event- driven Monitoring of Processes* ", http://docs.oasis-open.org/tamie/v1.0/200906/xtemp-1.0-csd01.pdf

Abbas Asosheh has received his B.Sc. in Communication Systems from Isfahan University of technology, Isfahan, Iran, in 1987. He received his M.Sc. in High Frequency Communication System from Sharif University of technology, Tehran, Iran, in 1991 and Ph.D. in Quality of Service Enhancement in Voice over IP Network from The School of Physical Science and Engineering, Kings' College London, UK, in 2005. He was the project manager in Iranian Research Institute for ICT (ex ITRC) Tehran, Iran from 1989 to 1994, the lecturer at Emam Hossein University of technical science, Tehran, Iran from 1988 to 1991, the lecturer at Sharif University of technology, Tehran, Iran from 1990 to 1991 and the lecturer at Tarbiat Modares University, Tehran, Iran, from 2006 till now. His research interests include Next Generation Network, Network Traffic Modeling, Wireless Network, Network Security, Service Oriented Architecture, Internet Data Centre, Distributed Enterprises and Intelligent Transportation Systems.

Pasha Vejdan Tamar received his B.Sc.degree in Computer Engineering from Electrical Engineering Faculty of Sharif University of Technology, Tehran, Iran. He also received his M.Sc. degree in Information Technology from Faculty of Engineering Tarbiat Modares University, Tehran, Iran. His research interests include e-business modeling and standardization, e-business testing & e-Strategy.