TESTBATN - A SCENARIO BASED TEST PLATFORM FOR CONFORMANCE AND INTEROPERABILITY TESTING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

TUNCAY NAMLI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

JUNE 2011

Approval of the thesis:

TESTBATN - A SCENARIO BASED TEST PLATFORM FOR CONFORMANCE AND INTEROPERABILITY TESTING

submitted by **TUNCAY NAMLI** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering**

Prof. Dr. Asuman Dogaç
Supervisor, **Computer Engineering Department, METU**

Examining Committee Members:

Prof. Dr. Özgür Ulusoy
Computer Engineering, Bilkent

Prof. Dr. Asuman Dogaç
Computer Enginering, METU

Prof. Dr. Hakkı Toroslu
Computer Enginering, METU

Prof. Dr. Nihan K. Çiçekli
Computer Engineering, METU

Assoc. Prof. Dr. Ahmet Coşar
Computer Enginering, METU

Date:

ABSTRACT

TESTBATN - A SCENARIO BASED TEST PLATFORM FOR CONFORMANCE AND INTEROPERABILITY TESTING

Namlı, Tuncay
Ph.D., Department of Computer Engineering
Supervisor : Prof. Dr. Asuman Dogaç

June 2011, 176 pages

Today, interoperability is the major challenge for e-Business and e-Government domains. The fundamental solution is the standardization in different levels of business-to-business interactions. However publishing standards alone are not enough to assure interoperability between products of different vendors. In this respect, testing and certification activities are very important to promote standard adoption, validate conformance and interoperability of the products and maintain correct information exchange. In e-Business collaborations, standards need to address different layers of interoperability stack; communication layer, business document layer and business process layer. Although there have been conformance and interoperability testing tools and initiatives for each one of these categories, there is currently no support for testing an integration of the above within a test scenario which is similar to real life use cases. Together with the integration of different layers of testing, testing process should be automated so that test case execution can be done at low cost, and repeated if required. In this theses, a highly adaptable and flexible Test Execution Model and a complementary XML based Test Description Language consisting of high level test constructs which can handle or simulate different parts or layers of the interoperability stack is designed. The computer interpretable test description language allow dynamic set up of test cases and provides flexibility to design, modify, maintain and extend the test functionality in contrast to a priori designed and hard coded test cases. The work presented in this thesis is a part of the TestBATN system supported by TUBITAK, TEYDEB Project No: 7070191.

Keywords: conformance testing, interoperability testing, test scripting, test automation

ÖZ

TESTBATN - SENARYO TABANLI UYGUNLUK VE B İRLİKTE İŞLERLİK TEST PLATFORMU

Namlı, Tuncay
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Asuman Doğaç

Haziran 2011, 176 sayfa

Günümüzde birlikte işlerlik kavramı e-Iş ve e-Devlet alanı için en önemli sorunlardan biridir. Bu probleme yönelik en önemli ve temel çözüm bilindiği gibi sistemler arası etkileşimlerin farklı seviyelerde standartlarının belirlenmesidir. Buna rağmen tek başına standartların belirlenmesi farklı şirketlerin ürünleri arasında birlikte işlerliğin sağlanması açısından yeterli olmayacaktır. Bu kapsamda test ve sertifikasyon aktiviteleri standartların benimsenmesinde, ürünlerin uygunluk ve birlikte işlerliğinin tasdik edilmesinde ve ürünler arası doğru veri akışı ve iş süreçlerinin sağlanmasında çok önemli yer tutar. e-Iş etkileşimlerinde birlikte işlenebilirliğin sağlanabilmesi için standartların iletişim, iş dokümanı ve iş süreçleri gibi faklı seviyelerde kısıtlamalar getirmesi gerekmektedir. Bu belirtilen her bir kısım için test araçları veya girişimler mevcut olmasına rağmen tüm bu kısımların bütünsel bir şekilde gerçek hayattaki bir süreci simule edecek bir test senaryosu kapsamında test edecek bir model mevcut degildir. Bu tez çalışması yüksek seviyede esnek ve değişik standartlara uyarlanacak şekilde Test Uygulama Modeli ve bu modeli tamamlayan XML tabanlı Test Tanımlama Dili tasarımını içermektedir. Test Tanımlama Dili bilgisayar tarafından işlenebilir olup test senaryolarının dinamik olarak tanımlanması, değiştirilmesi ve sürdürülebilmesini kolaylaştıracaktır.

Bu çalışma TUBITAK, TEYDEB programi tarafından desteklenen 7070191 numarali Test-BATN projesi kapsamiında gerçekleştirilmiştir.

Anahtar Kelimeler: uygunluk testleri, birlikte islerlik testleri, test otomasyonu

To my belowed uncle Ahmet Yılmaz...

ACKNOWLEDGMENTS

First of all, I would like to express my deepest gratitude to my supervisor Prof. Dr. Asuman Doğaç for all her guidance, encouragement, motivation and continous support throughout my graduate studies.

Special thanks to TestBATN development team Ali Anıl Sınacı, Günes Aluç, Atasay Gökkaya, Yiğit Boyar and my colleagues Gökçe Laleci, Yıldıray Kabak and Mustafa Yüksel at the SRDC Software Research & Development and Consultancy Ltd. for their cooperation and support they have provided.

I would also like to thank TUBITAK Teydeb for their support to the project “Birlikte Islerlik Standartlarinin Internet Tabanli Test Altyapisi (Internet Based B2B Test Framework)” where the TestBATN prototype has been developed and the reviewer of the project Oguz Dikenelli for his valuable guidance during the project.

Finally I would like to thank to my parents and my friends for their support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged [1]. More specifically, interoperability is said to exist between two applications when one application can accept data (including data in the form of a service request) from the other and perform the task in an appropriate and satisfactory manner (as judged by the user of the receiving system) without the need for extra operator intervention [2].

Interoperability is a major challenge of today's eBusiness and eGovernment applications. Several standards have been developed and some are still under development to address the various layers in the interoperability stack. Interoperability testing involves checking whether the applications conform to the standards so that they can interoperate with other conformant systems. Only through testing, correct information exchange among applications can be guaranteed and products can be certified.

To be interoperable, two applications need to agree on interfaces in each of the three layers of the interoperability stack: the communication and transport layer; the document layer which involves the format of the exchanged messages and documents as well as the coding systems used and the business process layer which involves the choreography of the interactions. Bilateral agreements between two applications on these interfaces are not practical because it requires the implementation of a new interface for each application to be communicated with. Therefore standards have been developed to address the various layers in the interoperability stack:

- Communication Layer: This layer covers the range from the transport and communication layer protocols like HTTP to the higher level messaging protocols. Furthermore, security, reliability and other quality of service protocols and extensions over the messaging protocols are also addressed in this layer. As an example, in eHealth for this layer, HL7 (Health Level 7) v3 [3] provides a number of normative transport specification profiles, namely, the ebXML Message Specification Profile [4], the Web Services Profile [5] and the TCP/IP based Minimal Lower Layer Protocol Profile [6].

- Document Layer: This layer addresses the content of messages exchanged among applications such as business documents (e.g. Invoice, Order) or Electronic Healthcare Records (EHR) in eHealth. For example, HL7 has defined standard message formats and an EHR standard, namely, HL7 Clinical Document Architecture (CDA) [58]. This layer also addresses the coding systems used in the messages, for example SNOMED CT [8], ICD-10 [62] or LOINC [10].

- Business Process Layer: This layer addresses the choreography of the application interactions, for example an acknowledgement message must be sent after receiving an "HL7 Admit Patient" message or "UBL Invoice" message must be sent after receiving an "UBL Order" in Universal Business Language (UBL) Norhtern European Subset (NES) [11] Simple Procurement Profile. The business process layer, either presented in a formal business process specification standard such as ebXML Business Process Specification Schema [12] or with an informal workflow definition like Integrated Healthcare Enterprise (IHE) interaction diagrams, provides a message cheoragraphy, exception flows (error handling) and the other business rules for the application roles participating in the process. For example, HL7 v3 provides storyboards which describe HL7 v3 message choreographies between specific roles in specific events [13].

Interoperability and conformance testing involves checking whether the applications conform to the standards so that they can interoperate with other conformant systems. Conformance testing, as the name implies, is about testing a single system to determine whether it conforms to an identified standard and hence it involves a system under test (SuT) and an application to test it. Interoperability testing, on the other hand, involves more than one system, each playing a different role in a given scenario and it is about testing their ability to exchange information and to use the information that has been exchanged. To realize the interoperability testing, testing frameworks must have the mechanisms to track the scenario and to capture the messages exchanged between the parties. Additionally, not all the roles that exist in a scenario may be present in the testing environment, and therefore when necessary, the testing framework must play the missing roles in the scenario by simulating them.

It is clear that, there are a variety of different standards that can be used for each layer of the interoperability stack and if applications conform to different standards, the interoperability problem continues. Profiling is used to overcome this challenge by predetermining the combination of the standards to be used and even further restricting them to ensure interoperability. For example, Integrated Healthcare Enterprise (IHE) [14] in eHealth and UBL NES in eBusiness are one of those initiatives which follow this profiling approach. Profiling concept and a methodology for reusing standard health information models is described in detail in [15]. Additionally, some governmental authorities like the HITSP in USA [16], the Ministry of Health, Turkey [17] and UBL NES in Europe published integration profiles for their national health systems and european public procurement networks.

It should be noted that whether it is an application conforming to a single standard addressing a specific layer in the interoperability stack or an application role in an interoperability profile which references several layers in the interoperability stack, without proper testing, the interoperability cannot be guaranteed. On the other hand, relying on testing each layer independently will not be sufficient to test the conformance and interoperbility. Testing for the conformance (or interoperability) of the sum is more than the sum of testing for the conformance of its parts [18]. There are testing tools and initiatives like National Institute of Standards and Technology (NIST) Testbed [19], KorBIT [20], WS-I Testbed [21] in eBusiness which addresses a specific standard (e.g. ebXML) and a specific layer in the interoperability stack. The telecommunication domain has also a very successfull test framework "TTCN-3 Testing and Test Control Notation" [22] published by European Telecommunication Standards Institute (ETSI) [23] which is used for the conformance and interoperability testing of telecommunication standards like Voice Over IP standards, WiMax (802.16) [24] and IP Multimedia Subsystem (IMS) [25]. In eHealth, the HL7 Message Maker [26], developed by the National Institute of Standards and Technology (NIST) and the Health Level 7 (HL7) Standards Consortium, addresses the messaging layer in the interoperability stack, that is, it is a tool for conformance testing of the HL7 messages. Although these tools are very helpful for the conformance testing of systems based on their specific target standards, when it comes to the new standardization approach, the profiling, which ensures the interoperability among systems for a specific real life scenario where individual standards is not able to, they are insufficient. Therefore, there is a need for a generic test framework which defines a standard representation of test cases and operational semantics that can perform conformance and interoperability tests for a whole business process specified by a set of standards even from different domains(e.g. IHE is using IT standards and eHealth standards in many of its profiles).

Being a generic framework is not the only characteristic that we seek for an effective conformance and interoperability test framework. There are futher points in conformance and interoperability testing that needs further improvement and which the existing test frameworks lacks partialy or completely. My study that will be described in this thesis aims to design a generic conformance and intereoperability framework, named as TestBATN, which enables:

- Responsible organizations to develop, deploy and maintain a test platform (tools, test cases, supplementary materials) for any eBusiness standard, specification or profile (based on a set of standards/specifications)
- Vendors to run the defined test cases anywhere and anytime to prove the conformance and interoperability of their products.

The objectives and principles of the TestBATN framework is as follows:

- A Generic Framework: Being a software framework is an important issue. Current eBusiness standards specify a variety of protocols, content format or choreographies. In order to support all of these and test them, the testbed should be adaptable and modular. Therefore, it is necessary to define interfaces for several layers and facilitate plug-in modules supporting different protocols or formats implementing the specified interfaces.
- Full Automation of Test Process: Partly automating the test process, that is, providing different tools for certain layers of the interoperability stack and doing the rest manually results in human labor intensive, error prone, costly to develop test processes. Automation of testing also implies the non-interference with the system in its native state of

execution. The TestBATN aims at testing the organizational interoperability and provide a holistic approach by involving configuration management and other preliminary test steps into the testing process.

- Easy Design: The testbed should aim at the "low cost of entry" for its users and hence provide a graphical environment where a test designer can assemble the reusable test constructs for conformance and interoperability tests.

- Easy Maintenance: Standard development is a continuous process and a new version may be published each year with some additions or updates on several parts of the standard. Furthermore, for certification or annual test events test cases need to be updated to prevent possible specific adjustments in the SuTs to pass the tests with some specific requirements. Therefore, TestBATN will provide easy maintenance for the test cases.

- Testing Anywhere, Anytime: At the age of Internet, interoperability and conformance testing should not be restricted in time and place. Vendors should be able to test their products over the Web anytime, anywhere and with any party willing to do so. In this way, not only the vendors, but also the customers who plan to buy or who already install the products in their premises can use such testing services.

- Detailed Reporting: One of the main objectives of the TestBATN in testing should be helping vendors to identify their errors, non-conforming, non-interoperable parts according to a standard. Therefore, the testbed should aim detailed reporting of each step in the testing process.

- Reduce Time Spent in Testing: Considering the amount of test cases to cover the conformance or interoperability testing requirements of a standard, it is self-evident that the time spent by participants during the testing process should be significantly reduced.

- Reusability and Testing Integrations: Currently, there have been conformance and interoperability testing tools and initiatives (e.g. NIST Business Document Content testbed, KorBIT ebMS testbed) for some specific standards addressing one or some layers of the interoperability stack. However, some initiatives which publish interoperability profiles covering a set of standards for different layers of the interoperability stack have been established. TestBATN aims to support integrated testing and enhance the reusability of test materials and the pluggable modules in integrated tests.

The TestBATN framework described in this thesis is realized within the scope of the project “Birlikte Islerlik Standartlarinin Internet Tabanli Test Altyapisi (Internet Based B2B Test Framework)” (Project No: 7070191) supported by TUBITAK Teydeb. The project was finalized in 2010 and the first prototype of the TestBATN framework [27] has been delivered. The first TestBATN prototype is used by the Ministry of Health(MoH), Turkey to perform conformance and interoperability tests on products of hospital information system vendors in Turkey against the National Health Information System specifications which are based on HL7 v3 standard. In this respect, the framework is used in two interoperability testing events of MoH one in Çesme and one in Ankara where all hospital information system vendors are tested by the test cases developed for TestBATN. TestBATN with the NHIS test scenarios has been also available from Web (http://www.srdc.com.tr/testbatn) since that time for the use of vendors to perform online testing. This study have also contributed to the CEN/ISSS Global Interoperability Testbed project - Phase I [28] supported by CEN. The project published the CEN Workshop Aggrement 16093:2010 - Feasibility Study for a Global eBusiness Interoperability TestBed (GITB) [29] where the use cases that is a part of my study takes part in. By the experience and knowledge gained from this study, the work is also presented and guidance is provided for interoperability testing in ETSI Special Task Force TR 370 - Standardisation of new methodology and framework for Automated Interoperability Testing of Distributed Systems [30] project.

The rest of this thesis is organized as follows: Chapter 2 briefly summarizes the existing test tools and frameworks. In Chapter 3, conformance and interoperability testing requirements of existing standars will be elaborated with two use case studies; HL7 v3 Use Case and e-Government-Public Procurement - CEN/BIIScenarios. HLv3 Use Case is detailed in [31] and e-Government-Public Procurement - CEN/BIIScenarios is detailed in [32]. The architecture of TestBATN framework is detailed in Chapter 4. In Chapter 5, utilization of TestBATN prototype in real life cases; Conformance and Interoperability Testing for Turkey’s NHIS Specifications [33] and Conformance and Interoperability Test Cases for IHE profiles [34] are summarized. Finally, Chapter 6 concludes the thesis and suggests possible future research directions.

CHAPTER 2

RELATED WORK

In this section, related works and frameworks regarding conformance and interoperability testing will be summarized. These existing testing frameworks or methodologies are mostly targeting a certain business domain or a specific standard.

2.1 TTCN-3: Testing and Test Control Notation v3

European Telecommunication Standards Institute (ETSI) [23] has developed testing methodologies for Conformance and Interoperability testing, and in particular the TTCN-3 Testing and Test Control Notation [22]. ETSI has more than 20 years of experience for applying robust test methodologies on Telecommunication networks. Telecommunication networks use many heterogenic protocols between multiple system nodes, with dedicated Configuration and Application data. Ensuring the interoperability of the components of such systems, across different vendor platforms and countries require testing the networks component accurately. For this purpose, ETSI has developed and used several means of test: i) Conformance testing for checking the compliance of isolated components of the Telecommunication networks, to the protocol standards, ii) Interoperability testing to verify the ability of network component from different vendors to operate in real conditions.

ETSI has provided and experienced methodologies for testing, which applies to all types of system and protocols, taking into account the heterogenic nature of the telecommunication networks used by ETSI members:

- An Implementation Statement Proforma (PICS), to present parameters of implementation of a protocol, and in particular which enable to identify and record the feature that are either mandatory, optional or conditional.,

- A test purpose framework to develop a Test description document, starting from a set of protocol standards, going through testing requirements,
- An abstract and efficient normalized Testing language: TTCN-3, to develop abstract test scripts, which enables dynamic configuration of Test systems, construction of any type of data and testing any simple or complex type of message choreographies.

TTCN-3 is a standardized test language suited to develop test cases for Conformance and Interoperability Testing for telecommunication networks and systems. It is a C like language that has some special constructs for conformance and interoperability testing. TTCN-3 is an ETSI standard, endorsed by ITU and universally recognized for testing telecommunication systems.

ETSI has published a set of standard to specify the TTCN-3 language as well as the integration of TTCN-3 in test systems and the mapping with existing notations:

- ETSI ES 201 873-1 TTCN-3 Core Language (CL)
- ETSI ES 201 873-2 TTCN-3 Tabular Presentation Format (TFT)
- ETSI ES 201 873-3 TTCN-3 Graphical Presentation Format (GFT)
- ETSI ES 201 873-4 TTCN-3 Semantics
- ETSI ES 201 873-5 TTCN-3 Runtime Interface (TRI)
- ETSI ES 201 873-6 TTCN-3 Control Interface (TCI)
- ETSI ES 201 873-7 using ASN.1 with TTCN-3
- ETSI ES 201 873-8 The IDL to TTCN-3 mapping
- ETSI ES 201 873-9 using XML schema with TTCN-3
- ETSI ES 201 873-10 TTCN-3 Documentation comment specification

Thus TTCN-3 does not only consist of a scripting language, but the TTCN-3 standard framework specifies also the interactions with the test system. This ensures the consistency of all different TTCN-3 compiler vendor platforms. The full specification of the Run time interface for instance offers a straightforward understanding and interpretation of the interaction between the TTCN-3 execution engine and the test system.The components of a TTCN-3 test system are as following: TE (TTCN-3 Executable), SA (System Adapter), PA (Platform Adapter), CD (Codec) and TM (Test Management). The communication between all parts of the TTCN-3 test system is fully specified in the above list of standards.

The main features of the TTCN-3 language are summarized by the 3C:

- Configuration: TTCN-3 enables the dynamic creation of several independent Test components and the dynamic establishment of communication ports between the test components and the System Under Test.

- Communication: TTCN-3 types suits to different types of communication messages either directly or through mapping. Then TTCN-3 message are sent or received enabling testing all kind of protocol process. The receive mechanism includes a very powerful verification of the messages sent by the system under test.

- Control: TTCN-3 includes a test execution control mechanism, which support a conditional execution of the test cases.

The TTCN-3 language has a well defined syntax, and includes a static and operational semantics. It follows a modular concept, enabling to split the definitions logically over several modules. This allows to share common definitions across several test suites by importing the required definitions from the common modules. The language is fully extendible via “attributes” or “external functions”. TTCN-3 uses a set of commands, which are fully dedicated to testing. This makes the language easy to use, and avoid defining functions to realize the testing event, because all necessary test command are already define. The 2 main testing commands are “send” and “receive”. Associated with some attributes, they enable to develop test sequences suited to all types of protocols. Furthermore, additional commands enable to control the synchronization of the send/receive event: like for instance timers or loop control. Finally, if / then / else constructs allows conditional.

TTCN-3 support usual language feature like subroutine, formal parameter, handling of variable with predefined and constructed types, etc. TTCN-3 supports the concept of “templates” of messages, which enables to create instance of messages to be sent, or expected to be received from the System Under Test. Message templates can be fully specified, or partially specified by using either wild cards or formal parameters in certain parts of the message. This wild card mechanism is required to match received messages when some field values are not predicable. Furthermore, TTCN-3 enable storing received messages in variable to re-use some field values in messages sent as a response back to the System Under Test.

Although, TTCN-3 is a computationally complete programming language for expressing test cases for conformance and interoperability testing in the telecommunication domain, it is not suitable to use in eBusiness and eGoverment domains. Due to the nature of testing problems in the telecommunications domain, TTCN focuses on the details of communication but is not able to fully represent business processes, exploting existing materials (e.g. XML schemas, schematrons) for testing. Furthermore, as it is a complex programming language it is difficult to understand and desing test scenarios for business experts.

2.2 OASIS Event-driven Test Scripting Model

The Event-driven Test Scripting Model (eTSM) is an ongoing initiative which aims to provide a model for integrated testing, and a scripting representation for the related test cases, with a particular focus on communication events, versatile usage for the quality assurance testing phase as well as the monitoring of deployed systems, and extensible design to leverage specialized validation processors as well as XML tools such as Xpath, XSLT and XQuery [18]. The test model includes advanced functions that support the test targets: business documents, message packaging, message choreographies and business transactions. These functions turn into high-level primitives of eTSM, which allow for concise expression of test cases. These functions include:

- Workflow control based on a thread model. This is embedded in the notion of Monitor, which is the basic execution unit for test cases.
- Event-driven scripts. The general control of test case execution within a test suite, and of the test suite itself is represented by Triggers which define under which conditions and events an execution takes place.
- Event logging and correlation. Event management, central to eTSM, is supported by an

entity called Event Board. The Event Board normally suffices for mediating all inputs to a test case, as well as outputs.

- Messaging gateways. Message traffic expected in all eBusiness applications, is mapped to and from events. Event-Adapters perform these mappings, allowing for abstracting test cases from communication protocol aspects.
- Semantic test plug-ins. Advanced verifications on business documents, ranging from schema validation to semantic rules over business content, are delegated to Eval-Adapters.

The eTSM script language is designed to ease the definition of testing or monitoring scripts as sequences of steps (flows), that may be synchronized by events as well as generate events. eTSM has extensive provisions for leveraging existing expression languages. One such language is XPath, which is the default dialect used in various expressions and conditions of eTSM, although other dialects are an alternative to be indicated in the language attribute of various eTSM statements. The following include a few examples of the general scripting units:

- Scriplet: Unit of script that have the following semantics: (a) scoping unit for variables (and generally scoping unit for variable assignment as well), (b) a scriplet can be invoked from other scriplets, with parameter passing, (c) a scriplet can invoke another scriplet "concurrently", meaning that both scriplets execute concurrently afterward. There is an implicit joining of all scriplets started from a (parent) scriplet: the parent only completes its execution after all children scriplets - either serial or concurrent - terminate.
- Function: The primary objective of functions is advanced computations that may exceed the capability of eTSM scripting. Functions can be invoked from many other statements including from inside expressions (e.g. an XPath expression), unlike scriplets. They may also be written in "foreign" languages or scripts, e.g. in Java, C-Sharp or XSLT, making it possible to reuse advanced function libraries and operators available for these languages, while scriplets are exclusively coded in eTSM syntax. A function returns a value that could be an XML fragment. This value may be typed, and by default is a string possibly empty. Another difference between scriplet and function, is that the effect of executing a function is all in its returned result. There is no other side-

effect observable in eTSM (besides any intentional event posting done by the function). In contrast, a scriplet does not explicitly return a result. However, its execution produces an "effect" which is the concatenation of the effect of each step inside it. Thus, a scriplet execution could compose an XML fragment as its effect. Assigning a scriplet execution to an eTSM variable leads to assigning the effect of its execution as value of the variable.

- Catch: Selection of one or more events from the event board within a time window. The selection is based on an XPath expression or an XQuery. The operation may wait for the event(s), acting as a synchronizing control. The same catch operator is used in two different contexts of event catching: (1) Live execution, where events may not have occurred yet. In that case, catch is a synchronization operator that will cause a script execution to wait for the expected event(s). (2) Stale execution, where events have already occurred and are logged in the event board. In that case, catch acts as a query to the event board. There are a few more operators being developed under OASIS TaMIE Technical Committee. Those include operators related to iterations, conditional statements, event posting and management. Although eTSM is under development, it will impact to global test bed development due to the following reasons. First, eTSM intends to minimize customized development of a test bed, implying that the test framework may include event board and manager, and execution mechanism of monitors. Second, eTSM scripts can be converted into XSLT whose compiling and executing tools are commercially available.

The eTSM execution model is shown in Figure 2.1 and it consists of the following components:

- Monitor: Represents the logic of a test case. A test case may use several monitors in its definition, and a test case instance may engage the concurrent or sequential execution of several monitor instances. A monitor is a script that specifies the steps and workflow of the test case. A monitor instance is always created as the result of a start operation executed either by another monitor or by a trigger. There is always a trigger at the origin of monitor(s) execution. A monitor instance can start another monitor instance concurrently to its own execution, and can activate another trigger. The outcome of a test case (pass / fail / undetermined) is determined by the final outcome of the monitor(s)

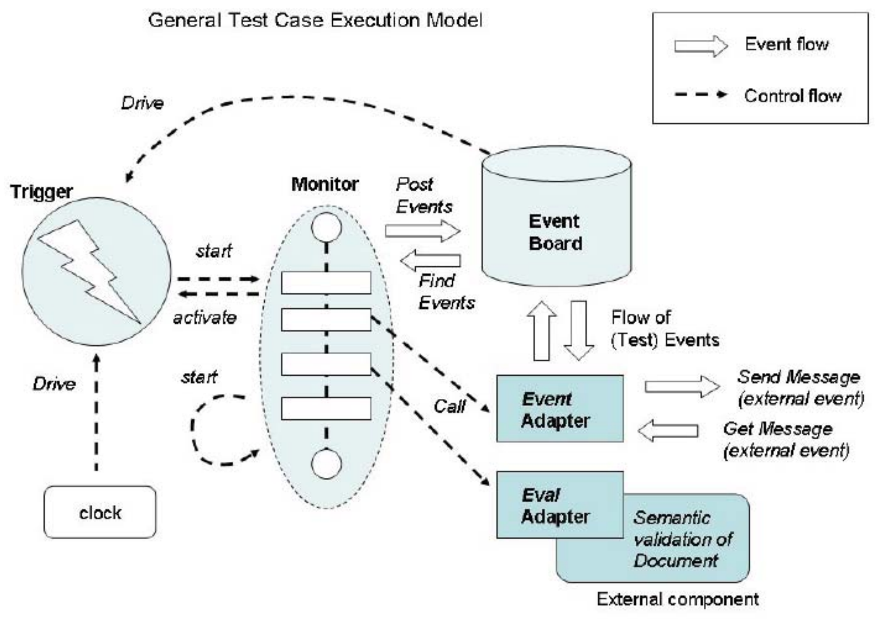


Figure 2.1: eTSM Test Execution Model

implementing this test case. The execution of a monitor produces a trace that can be posted as an event.

- Trigger: The trigger is a script that defines the event or condition that initiates the execution of the test case, i.e. the execution of a monitor. A trigger can be set to react to an event (event-watching) or to a date (clock-watching), and is associated with one or more monitors. Because a trigger initiates the execution of a test case, it is usually not considered as part of the test case itself, but part of the test suite that coordinates the execution of several test cases. A trigger is active when ready to react to events for which it has been set, and ready to trigger its associated monitors. When a trigger starts a test case, a case execution space (CES) is allocated, within which the created monitor instance as well as all subsequent dependent instances will execute. The CES defines a single scope of access to events and to other objects referred to by variables. When activated, a trigger is given a context object, which will be part of the CES of the monitor(s) the trigger will start.

- Test Suite: A test suite is a set of test cases, the execution of which is coordinated in some way. This coordination may be represented by a monitor, that will either directly start the monitors that represent individual test cases, or that will instead activate triggers that control these monitors. For example, a test suite may serialize the execution of test cases TC1 and TC2 by setting a trigger for TC2 that reacts to the event posted by TC1 at the end of its execution. Or, the test suite may set a trigger that will initiate the concurrent execution of several test cases.

- Event: An event is a time stamped object that is managed by the Event Board. Events are used to coordinate the execution of a test case, and to communicate with external entities. For example an event may serve as a triggering mechanism (in event-driven triggers) for test cases, as a synchronization mechanism (e.g. a test step waiting for an event) or as a proxy for business messages, in which case the mapping between the event representation and the business message is done by an event adapter. Some events are temporary, which means they are only visible to monitors from the same test case execution (CES) and are automatically removed from the EB at the end of the CES they are associated with. Event Board (EB): Decisions about the outcome of a test case will often require access to a history of past events. Such a history of events is abstracted here as the "Event Board (EB)". Relying on event analysis for test cases also provides more control on the execution time, which may be deferred. Events can be posted to the board, or searched. An event board can be seen as an event log that supports additional management functions. The event board is the main component with which a monitor interacts during its execution and in many cases the only one. The Event Board also provides operations for managing events. A standard (XML infoset) representation of the invocations of such operations is proposed here, for the purpose of portability of test case scripts, and leaves the API aspects and its details, to the implementation work.

- Event Adapter: An event adapter is a mediator between the external world and the event board. It maps external events such as message sending/receiving, to test events and vice versa. For example, an event adapter will interface with an eBusiness gateway so that it will convert received business messages into a test event and post it on the event board. Conversely, some events posted on the event board by a monitor can be automatically converted by the adapter into business messages submitted for sending. An event adapter can also be directly invoked by a monitor. Whether the adapter is designed to react to the posting of an event on the board or is directly invoked by the monitor, is an implementation choice. In both cases, it would convert a test event into an external action.

- Eval Adapter: An eval adapter is implementing or interfacing with an implementation of a test predicate that requires specific processing of provided inputs that is not supported by the script language. Typically, it supports a validation check, e.g. semantic validation of a business document. An eval adapter is always invoked by a monitor. On invocation, an eval adapter returns an XML infoset summarizing the outcome, which can be evaluated later in the monitor workflow. Document processors fit in this category: Schematron, XSLT, OWL Reasoner.

2.3 NIST Application Information Mapping Test Bed

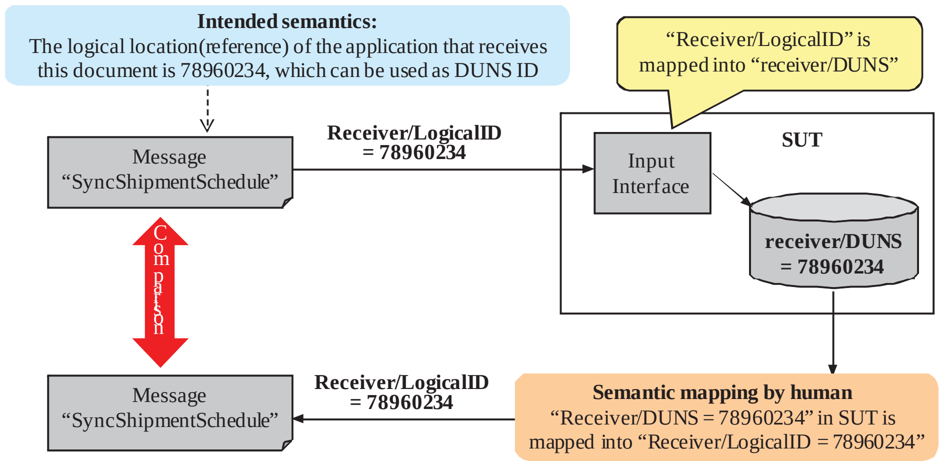

Intended semantics:
The logical location(reference) of the application that receives this document is 78960234, which can be used as DUNS ID
"Receiver/LogicalID" is mapped into "receiver/DUNS"
Message "SyncShipmentSchedule"
Receiver/LogicalID = 78960234
Input Interface
SUT
receiver/DUNS = 78960234
Comparison
Message "SyncShipmentSchedule"
Receiver/LogicalID = 78960234
Semantic mapping by human
"Receiver/DUNS = 78960234" in SUT is mapped into "Receiver/LogicalID = 78960234"

Figure 2.2: General procedure of NIST input test

An information-mapping test can have two parts: an input test and an output test. The input test verifies that the system under test (SUT) successfully reads and interprets the content in the standard data exchange representation into its local representation. In the "Comparison" step, the output from the test is compared to the input to determine if the instance data are in fact semantically equivalent after the translation by the input interface. Figure 2.2 illustrates the procedure for the input test. This general testing procedure is often impractical, because the test bed may not directly communicate with the SUT which is supposed to be a part of enterprise's internal system. However, if we were to perform the tests following these general procedures, unique test data and test result analyses would be required for the SUT interface. For this reason, a testing procedure should run only on a single representation. The procedure relies on a human to interpret the test data in the proprietary representation back to the standard representation. Thus, semantic equivalence in the test result analysis is performed between instances in the same standard representation.

Conversely, the output test verifies that the application interface successfully reads and interprets the content in local representation into the standard representation. The output test requires the test data in a proprietary (non-standard format) but the result be analyzed in the target standard representation, i.e., the standard instance. This is opposite to the case of the input test. In this case, the human is used to interpret the data from the standard representation into a proprietary representation. The test data can be maintained in a single representation and the test-result analysis module needs to understand only a single representation.

Depending on the role an application plays in the integration scenario, it may require one or both tests. For example, a part ordering system which may generate purchase orders but may never consume them can only be tested for the output test. If a SUT has both the input and output interfaces, the only one test needs to follow this semi-automated procedure; the other can be fully automated. The precondition is that the first interface tested using the semi-automated procedure has to be free of incorrectness before performing the other automated procedure. This precondition is necessary to prevent the coincidental correctness resulting from symmetric mapping. That is if the input test is performed semi-automatically and debugged, then the output test can be fully automated. In this procedure, the output interface is tested by first providing the test data through the input interface that, then, triggers the SUT to post-process the same data through the output interface.

The test case generation and execution tool has been implemented in a “mapping test bed”. The test bed has been developed as an extension to the ebXML Interoperability, Implementation, and Conformance (IIC) test framework specification. The test user first uploads the MappingProfile table to the mapping test bed, and then the test bed generates test requirements and corresponding mapping test cases and document instances. In the second step, the test user selects a mapping test case to execute and the test bed transforms the mapping test case into an executable test case that is a set of procedural instructions/steps understood by the IIC test driver engine. The test driver engine follows the instruction to send, receive, and analyze test messages to generate test results.

The structure of mapping test case is similar to that of the IIC test case; however, the mapping test case is independent of the underlying communication protocols. A mapping test case consists of the following test steps.

- PutMessage is an instruction to construct and send the document instance. It has a Uniform Resource Identifier (URI) pointing to the instance file. There are two possible modes associated with the PutMessage instruction: automatic and manual. The automatic PutMessage initiates the test bed to send the document instance to the SUT on the wire, while the manual PutMessage initiates the test bed to display the instance to the test user via the web browser interface.

- GetMessage is a query instruction to retrieve from the “message store” the document instance message received from the SUT. Similarly there are automatic and manual modes. The automatic GetMessage invokes the test bed to retrieve the instance among the messages that are received on the wire, while the manual GetMessage invokes the test bed to retrieve the instance among those input by the test user via the web browser interface.

- TestAssertion is a container for validation statements. It contains the XPATH and XQuery scripts automatically generated from the test document instance. For each test requirement, the test bed generates test instances, in which test data are the values for the elements/attributes. Test data are generated automatically according to the test data meta-data. The user has to provide inputs on the elements/attributes where the test data meta-data is indicated as “user specified”.

To execute a test case, the test bed transforms it into an executable IIC test case that is defined as a sequence of operations (test steps) over the test harness. Each test step includes a transport and communication protocol configuration material (e.g., the ebXML Collaboration Protocol Agreement) and test verification conditions that define criteria for passing this test step. For the IIC executable test case generation, the test bed requires the following information: communication protocol, end points and operation name, packaging structure (e.g., permitted document schema and enveloping constraints), security configuration, and required messaging behavior (e.g., SOAP faults rules, acknowledgements, and sync/async messaging).

In addition to functional capabilities in testing, the NIST Application Information Mapping Testbed (AIMT) provides the following non-functional aspects:

- Optimized test case generation: The AIMT has a basic test requirement which is “All message elements should be instantiated at least once in standard messages.” The test requirement implies that each element of the standard message needs to be checked whether its semantic meaning is preserved in the applications or not. Repetitive verifications with different values may not be necessary since the test is only concerned with the semantic equivalence between message elements and data elements in the SUT. Basically, one instance can contain all the message elements if there is no particular restriction in message specification. In such an ideal case, the input/output test with one test case is sufficient to ensure no mapping error. In real cases, however, multiple instances should be populated not to violate message restrictions that exist in message specification. The minimization of test cases for the input/output test is important in real practice since the human involvement step may force the tester to spend much time on conducting the tests. The AIMT provides an optimization methodology to generate test cases which is to minimize the number of test cases while ensuring the basic test requirement.

- Message-independent test harness: The AIMT supports a message-independent testing using the Message Metamodel proposed by National Institute of Standards and Technology (NIST). The NIST Message Metamodel is a neutral form for representing messages of all these syntactic varieties, which excludes the specific representation rules for specific syntaxes. The Message Metamodel that holds message contents in syntax-neutral form enables the AIMT to deal with the message contents regardless of the chosen message syntax. In addition, test harness based on the Message Metamodel can be reused without regard to message-representation syntaxes.

- Support for manual steps: In order to enable the equality checking with input and output message instances, a business analyst as tester maps the message content from its local representation to the standard message-representation for the input test, or vice-versa for the output test. However, it is difficult for human to interpret the semantic meaning of the elements in the standard message-representation. Thus, the input test of AIMT provides a Narrative and questionnaire interface, where business analysts answer the questions that assess values of business domain entities and their elements (rather than to request values of the standardized document elements). The output test also provides the business analysts with the Narrative and value interface to capture the semantics of the business domain entities and their elements and to correctly map the message elements into the data elements in the SUT without the requirement to understand the standard message format.

2.4 NIST Business Document Content Test Bed

The XML Schema Quality of Design Tool(QOD), also called the QOD Tool, assists in consistently using XML Schema for the specification of information. Consistent design of XML schemas within an organization or single integration project can reduce the number and the severity of interoperability problems. In addition, this consistency makes the XML schema easier to extend, understand, implement, and maintain; and, it paves the way for automated testing and mapping. Applying best practices is one way to achieve this design consistency.

The purpose of QOD is to provide a collaborative environment for checking XML schema design quality. QOD is intended for both people developing guidelines for writing high quality XML schemas and those writing XML schemas. The system provides a database of guidelines, as well as, tests for checking for adherence to the guidelines. The guidelines come from a number of sources including documents from a variety of standards organizations. Users also may add guidelines and tests to the system.

The QOD covers following functional areas of Naming and Design Rule(NDR) development: authoring and testing. A particular emphasis is given to testing and sharing areas, while interfaces to 3rd party tools will be provided to support the NDR authoring process. The authoring component addresses integrating NDR authoring capabilities with a testing environment. Executable tests can be managed in conjunction with the NDR documentation. It also includes mechanisms for rendering NDR documentation in various formats, such as spreadsheets, PDF files, and editable markup using standard Docbook XML. In addition, the authoring capability will include features to support the generation of NDR documentation from rules available in the database. The authoring environment is discussed in detail in NDR Profile Schema Version 1.0 User Guide. Representation addresses how the information is captured in a structured way so that it can be more easily processed by a software application. Presentation refers to various ways in which information can be displayed to an end user. Editing addresses the process of changing the underlying information. In this environment, since the NDR itself is marked up using XML, much of the authoring capabilities rely on generic capabilities of that technology.

The foundation of the NDR Authoring capabilities is the NDRProfile schema. The NDR-Profile schema captures the information needed to support testing and sharing in a structured way such that it can be presented in multiple formats and integrated with a broader set of documentation. Several of the presentation tools developed for the authoring demo have been made available for others who would like to tailor presentation of the NDR. These include XSLT scripts for rendering the NDR in various formats including spreadsheets, PDF files, and editable mark up using DocBook. See the NDR Profile Schema Version 1.0 User Guide for more details. Editing of an NDR document has typically been done in a traditional word processing system. The document then needs to be tagged to work with the NDRProfile schema. However, by using an XML native editor with the NDRProfile schema this tagging step can be by-passed. The NDR developer(s) have the ability to author the NDRProfile in any tool appropriate to their environment. NDRProfile instances which are created using a native XML validation tool can be imported directly into the QOD. Authors can create their NDRProfile using non-XML software. Other templates (current and proposed) for standard software, such as Microsoft Word and Excel, along with transformation scripts (XSLT) will provide the author the ability to convert non-XML NDR's to NDRProfile XML for importing into QOD.

The primary use cases for testing considered in the design for QOD is stand-alone testing, where a schema developer wishes to test their schemas against a single set of rules and is not interested in sharing the results of the tests. This scenario may be supported by a stand-alone system that does not depend on connections to the Internet. Stand-alone testing is done by using an instance of an NDRProfile. The NDRProfile contains both the text of the NDR document as well as test scripts. The document can be rendered in multiple formats for viewing as described in the authoring section, and the test scripts can be extracted for execution against an XML schema. The testing features relate to the encoding of the rules from an NDR in such a way that an XML schema can be tested for compliance to those rules. In QOD, NDR rules can be encoded using Schematron or JESS. In the future, support for others of the emerging W3C standards such as XQuery may also be included as needed. The test scripts can be executed on a schema using 3rd party tools since the scripts are written using well-supported languages.

2.5 KORBIT Testbed

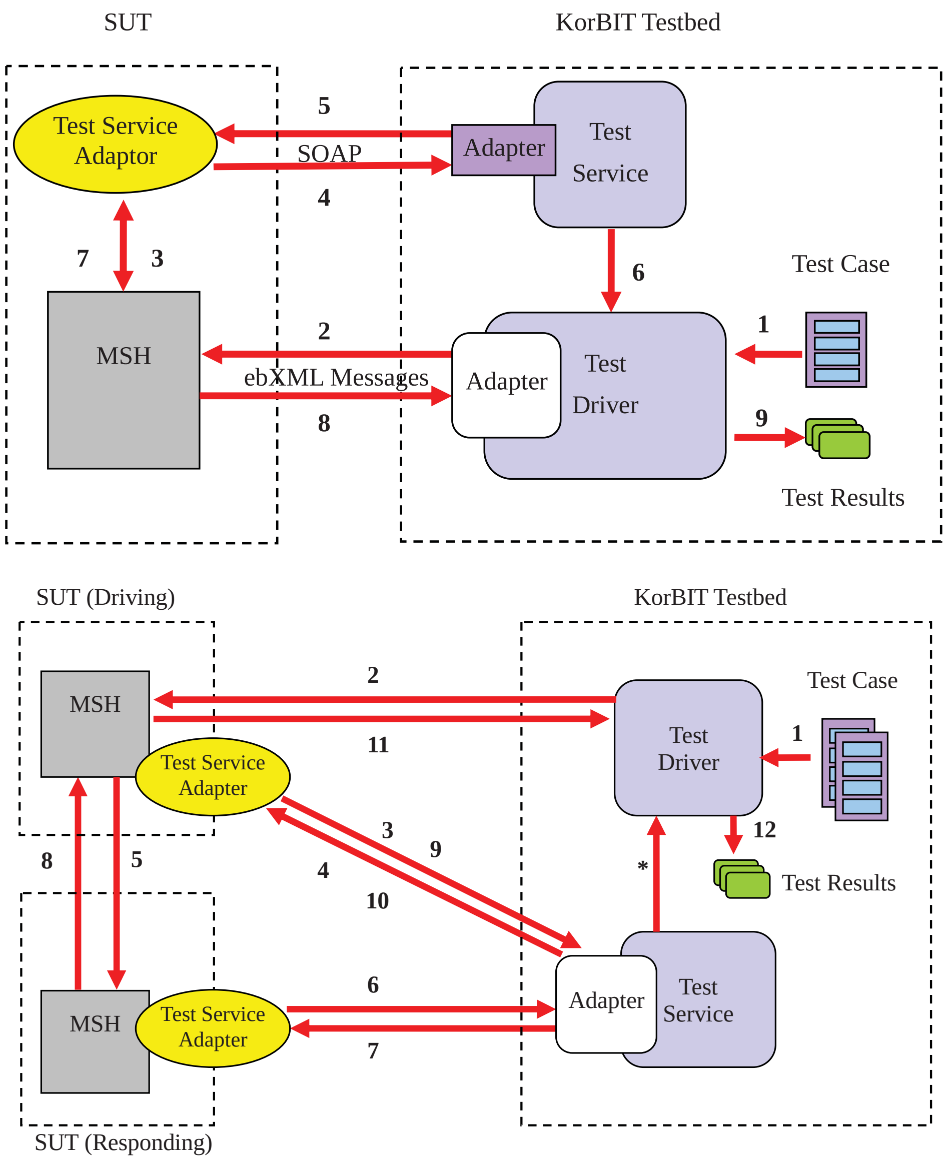


Figure 2.3: KORBIT Conformance and Interoperability Testing Setups

KorBIT was founded to help enterprises rapidly and effectively adopt eBusiness solutions.

In particular, KorBIT developed a testing system (KorBIT test bed) for evaluating and validating the implementations of eBusiness standard specifications such as ebXML Messaging Service (ebMS 2.0 and 3.0), UNeDocs guidelines, and Korea Institute EBusiness Commerce (KIEC) document guidelines. It supports both conformance and interoperability testing. The purpose of the Conformance Test is to verify whether a candidate MSH implementation is developed in conforming to the ebMS specification. The purpose of the Interoperability Test is to verify whether two candidate MSH implementations communicate without any errors according to a specific subset of the ebMS 2.0 specification. The KorBIT test bed 1.0 has been developed according to the ebXML IIC Test Framework v1.1, which supports conformance and interoperability testing for ebMS specification v2.0. Recently several modifications and extensions have been made as KorBIT test bed 2.0 according to event-driven test script modeling language being developed by OASIS TaMIE Technical Committee. KorBIT test bed 2.0 provides an infrastructure platform on which different types of test modules can be plugged in, such as ebXML messaging test module, web service messaging test module, document schema validation test module.

For instance, the test procedure for ebXML messaging test in the KorBIT Test bed consists of two phases. The first phase involves online conformance testing for individual ebMS solutions (hereafter, we use “MSH” to indicate an ebMS solution), while the second phase focuses on online interoperability testing between any pairs of MSHs. Conformance testing consists of the 7 different testing functions as follows: Message Packaging, Core Extension Elements, Error Handling, Sync Reply, Reliable Messaging, Message Ordering, and Security. Interoperability testing consists of the 13 different testing areas: Basic exchange without payload, One way messaging, Basic asynchronous exchange with one payload, Basic asynchronous exchange with two payload, Basic asynchronous exchange with three payload, Test unsigned AckRequested message with unsigned Acknowledgment, Process multipart/no payload messages, Process unrecognized MIME headers, Not understood SOAP Header extension elements are rejected, Not understood SOAP Header extension elements are ignored, Basic exchange with Error Action Message, Basic synchronous exchange with one payload, Test Synchronous unsigned AckRequested message with unsigned Acknowledgment, The KorBIT test bed is easy to use: the user has only to develop a test service adapter to convert the format of a message between a system under test (SUT) and the Test Service.

The KorBIT Test Bed evolved version 1.0 to 2.0, that is, from ebXML messaging test to including various B2B standard protocols such as web service messaging, process specification, and document contents testing. Event driven test script language (ETSL) enables KorBIT Test Bed 2.0 by providing the basis for representing various test cases in a single format. The ETSL Processor interacts with the event manager to manage events incoming from external entities, such as SUT. By doing so, various B2B standard protocols can be accommodated via "Test Module Adapter". The Test Module Adapter should be developed for every test module.

Figure 2.2 illustrates the conformance and interoperability testing configurations. In the configuration, the Test Driver initiates the testing and stores the test results. That is, the Test Driver verifies test conditions and generates test outputs, even if the Test Harness involves several (possibly remote) components. Significant events occurring in such components are sent back to the Test Driver.

The Interoperability Test Harness is used for verifying whether two SUTs communicate with each other without any interruptions. The Test Driver verifies test assertions and generates test results. Significant events occurring in such components are sent back to the Test Driver. The Interoperability Test Procedure will be asymmetric: one party will drive the Test Cases, and the other party will respond to the messages.This Test Harness can execute partial interoperability test, because only the driver party can control the Test Suite and test messages. In order to achieve a complete interoperability test between two parties, both parties must alternately play the driver role and the responder role.

2.6 WS-I Test Bed

The Web Services Interoperability Organization (WS-I) is an open industry organization chartered to establish Best Practices for Web Services interoperability, for selected groups of Web Services standards, across platforms, operating systems and programming languages. WS-I comprises a diverse community of Web Services leaders from a wide range of companies and standards development organizations (SDOs). The WS-I committees and working groups create Profiles and supporting Testing Tools based on Best Practices for selected sets of Web Services standards. In August 2003, the WS-I published the Basic Profile 1.0. This profile contains implementation guidelines for the core Web services specifications: XML 1.0, XML Schema 1.0, SOAP 1.1, WSDL 1.1 and UDDI 2.0. These guidelines are a set of requirements

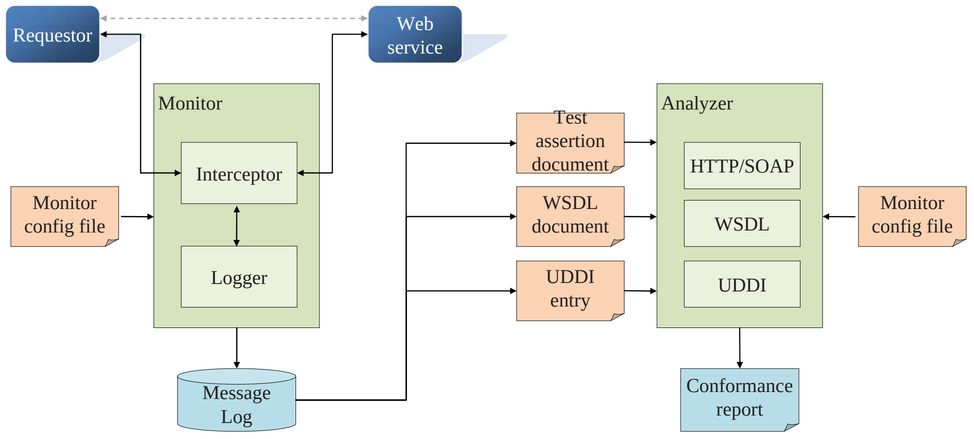


Figure 2.4: WS-I test tools architecture

that define how these specifications should be used to develop interoperable Web services. The WS-I test tools can be used to verify that a Web service conforms to these requirements. Tests are self-administered and aim to discover unconventional or faulty specification implementations and improve interoperability among applications and across platforms. The test results will help companies ensure that their web services are compliant with interoperability guidelines.

The WS-I Test Tools consist of two tools: the monitor and analyzer. This monitor captures messages and stores them for later analysis. The monitor itself captures messages traveling over different protocols and transports. The first version of this tool focus on being able to accurately capture HTTP based SOAP messages. Also, while many interception techniques are available, this implementation uses a man in the middle approach to intercept and record messages. The analyzer is used to validate that the Web service interactions contained in the message log conform to a WS-I profile, WS-I Basic Profile 1.0 for the first version. The purpose of the analyzer is to determine if a set of Web service related artifacts conform to the requirements in the WS-I Basic Profile 1.0. There are three basic types of artifacts:

- messages – The set of messages that were logged by the monitor.
- description – The service description for the Web service (this includes any referenced XML schema definitions), if the location of the WSDL document is available.
- discovery – The UDDI entries for a Web service, if the UDDI entries reference a

WSDL-based service description.

Figure 2.4 provides an overview of the WS-I Test Tools architecture. The monitor contains two primary functions: message interceptor and message logger. The message interceptor intercepts the messages that are sent from a service requestor to a Web service and from a Web service back to the service requestor. The logger formats the intercepted messages into a standard format and then writes them out to a message log. With these two functions, a single monitor can intercept and log messages from multiple Web services. The monitor functions are controlled by a configuration file which defines the association between the ports the monitor listens on for incoming messages, and the Web service location where the monitor should forward the messages.

When using the monitor, the service requestor views it as if it was the Web service. All SOAP messages are sent to the monitor instead of the Web service. Since this is not the normal mode of operation for the requestor, there are three basic ways to do this:

- Alter the requestor to point at the monitor instead of the Web service.
- Move the Web service to a new location and run the monitor in its place.
- Alter the Web service endpoint information that the requestor uses.

There are several system configurations that can be used to run the monitor. There are four basic system configurations which define the systems where the requestor, monitor, and Web service can run.

- The requestor, monitor, and Web service can be run on the same system.
- The monitor can be run on the same system as the requestor, and the Web service can run on a different system.
- The requestor can be on a different system than the monitor and Web service.
- The requestor, monitor, and Web service can be run on three different systems.

The analyzer tool determines if the artifacts for a Web service conform to the Basic Profile by processing a set of test assertions. A test assertion is a testable expression of one or more requirements in the Basic Profile. All of the test assertions are listed in a test assertion document, which is an XML document whose contents are segmented by artifact type (discovery, description, and messages). The input for the analyzer includes the location of the test assertion document and references to the Web service artifacts. All of this information is specified in the analyzer configuration file. The output from the analyzer is a conformance report.

CHAPTER 3

DOMAIN ANALYSIS FOR CONFORMANCE AND INTEROPERABILITY TESTING

In order to identify the basic requirements of a generic B2B conformance and interoperability testing environment, at the early stages of this study, conformance and interoperability constraints of two major standards HL7 v3 in eHealth domain and UBL/NES in eBusiness are analysed in detail. Use case approach is applied in this analysis and this chapter will summarize the summary of the results. The aim is to identify the common conformance constraints and approach so that the common testing requirements in terms of functionality can be identified.

3.1 Health Level (HL7) v3 Use Case

One of the most important family of standards in eHealth is Health Level Seven (HL7). In fact, HL7 v2.x is the most widely used eHealth message standard in the world today. However, being HL7 Version 2 compliant does not imply direct interoperability between healthcare systems. This stems from the fact that Version 2 messages have no explicit information model, rather vague definitions for many data fields and contain many optional fields. This optionality provides great flexibility, but necessitates detailed bilateral agreements among the healthcare systems to achieve interoperability. To remedy this problem, HL7 Version 3 is developed, which is based on an object-oriented data model, called the Reference Information Model (RIM).

With HL7 V3, conformance is specified in terms of application roles which are abstractions that express a portion of the messaging behavior of an information system. A vendor of a healthcare application describes its conformance by asserting that it completely supports all trigger events, messages and data elements associated with one or more application roles. This level of specificity allows clearer pre-contractual understanding between vendors and buyers and serves as a basis for conformance testing.

“The Health Level Seven (HL7) version 3 (v3) standards is a specification that defines static and dynamic models for messaging among healthcare applications. The static models define the structure of message content and the dynamic models define the expected behavior of the sending and receiving components.” The conformance to the standard is based on Application Role concept. In HL7, an application role is an abstraction that defines a portion of the messaging behavior of an information system. However, the Application Role concept should not be interpreted as the specification of complete behavior of a specific healthcare application. It describes only the messaging interface for the application for a specific messaging behavior. A healthcare information system may claim conformance to several or many application roles.

The structure and potential behavior of messages from a sending application role to a receiving application role are defined by interactions. In the HL7 v3 standard, the Interaction concept is formally defined as follows: “A unique association between a specific message type (information transfer), a particular trigger event that initiates or ”triggers” the transfer, and the Receiver Responsibilities (in terms of response interactions) associated with the receipt of the Interaction.”

The “Message Type” is simply the definition of the structure and semantics (the meaning of each message element) of the information transfer for a specific messaging. HL7 has a formal methodology for developing message types starting from its Reference Information Model (HL7 RIM) by using HL7 Vocabulary and HL7 Data Types. However, in the scope of conformance and interoperability testing, we can concentrate on the final product, the Message Type (or Hierarchical Message Descriptions) definitions which define a unique set of constraints for the message payload of an interaction.

“Trigger Event” is another crucial concept in HL7 for conformance and interoperability. It is actually the explicit set of conditions that initiate the transfer of information between application roles. HL7 defines one or more trigger events for each HL7 interaction. In terms of conformance, the application roles are expected to initiate the corresponding interaction when a trigger event occurs. In the HL7 v3 standard, the trigger events can be in the following types:

- Interaction Based: Trigger events can be based on another interaction. For example, the response to a query (which is an interaction) is an Interaction Based trigger event.
- State-Transition Based: HL7 defines states for some classes in its information model. These classes are called subject classes and played a central subject role for the interactions. HL7 defines state diagrams for such classes and defines state-transitions based on trigger events. For example, fulfillment of an Radiology Image Order in an application may change the state of the order from active to completed. State-transition based trigger events are resulting from a state transition for a class.
- User Request Based: Trigger events may be based on a user request. For example, a user can press a button in a hospital information system user interface to query medical records of a patient.

The responsibility of a receiver application role after an interaction can be one of the following:

- To initiate a trigger event which in turn can initiate a response interaction, or
- To initiate an interaction with another application role or
- To result in a state-transition change.

Additionally, the responsibility can be to perform the necessary modifications on the internal state of the application role so that the application role responds correctly to the future interactions. For example, a health record repository should store the electronic healthcare record (EHR) transmitted by an interaction so that the applications can reach the record by future interactions. This type of responsibility causes a more complex situation for conformance and interoperability testing. HL7 defines such situations with so-called Storyboards which describes real life scenarios among two or more application roles. The storyboards are the informative part of the standard and described by a sequence diagram and narrative text. Although they are not related with conformance, they illustrate real life scenarios and the choreography among application roles.

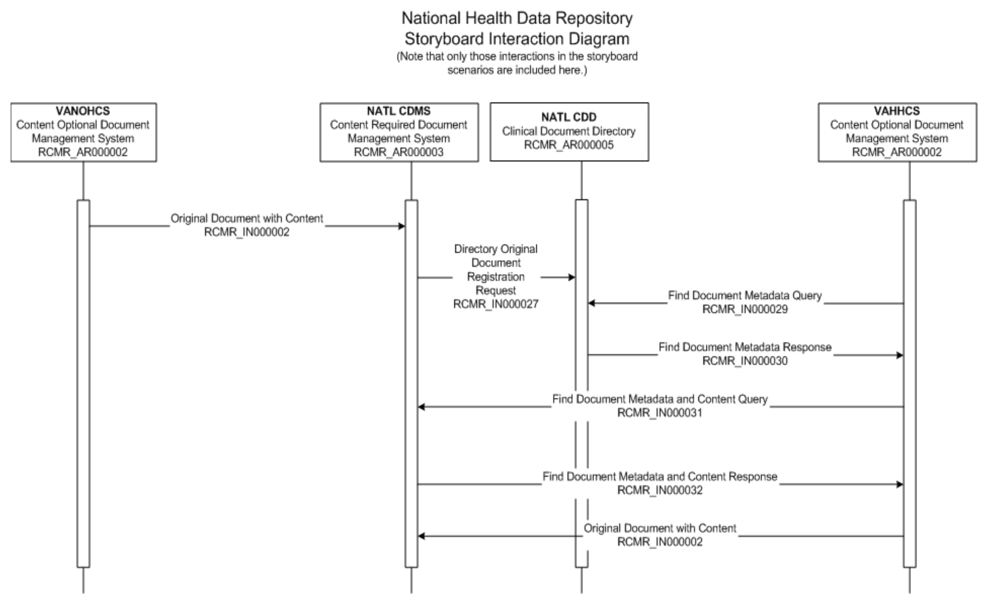

National Health Data Repository
Storyboard Interaction Diagram
(Note that only those interactions in the storyboard scenarios are included here.)
VANOHCS
Content Optional Document Management System
RCMR_AR000002
NATL CDMS
Content Required Document Management System
RCMR_AR000003
NATL CDD
Clinical Document Directory
RCMR_AR000005
VAHHCS
Content Optional Document Management System
RCMR_AR000002
Original Document with Content
RCMR_IN000002
Directory Original Document Registration Request
RCMR_IN000027
Find Document Metadata Query
RCMR_IN000029
Find Document Metadata Response
RCMR_IN000030
Find Document Metadata and Content Query
RCMR_IN000031
Find Document Metadata and Content Response
RCMR_IN000032
Original Document with Content
RCMR_IN000002

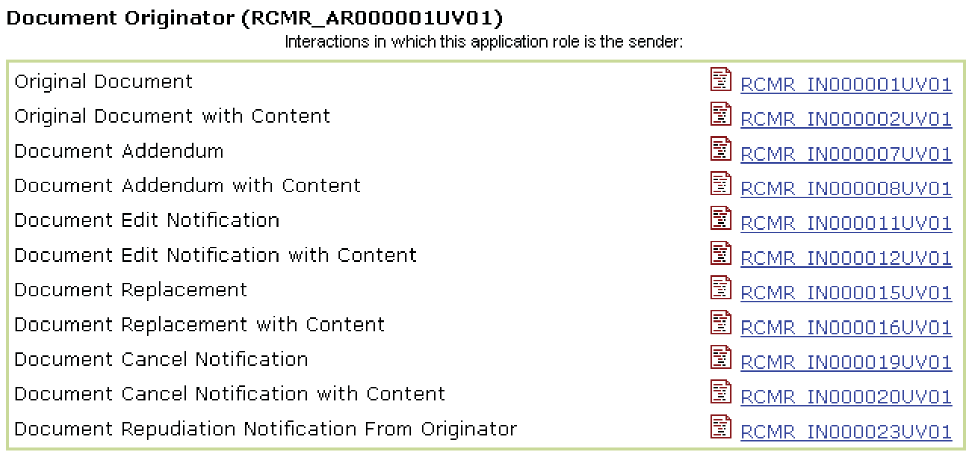
Figure 3.1: An example HL7 storyboard

The Figure 3.1 illustrates a storyboard defined in HL7 v3 standard regarding the management of clinical documents. Content Optional Document Management System, Content Required Document Management System and Clinical Document Directory are the application roles. For example, the Content Optional Document Management System is an abstract role for a medical information system which can generate clinical documents. Original Document with Content is one of the interactions in the storyboard and described by HL7 by its corresponding Trigger Event (Original Document Notification) and Message Type (Document Event, with Content). Through this interaction, the system playing the Content Optional Document Management System role stores the clinical document into a shared repository. As the receiver responsibility, the Content Required Document Management System should initiate the Original Document Notification trigger event which then initiates the Directory Original Document Registration interaction.

In summary, a system that wants to make a conformance claim to the HL7 v3 standard should first identify the application roles to which it claims conformance. For these application roles, the HL7 V3 specification directly states: i) The trigger events the system shall recognize, ii) The messages that the system shall send in response to trigger events or other messages, iii) The data content of these messages. The specification also states the messages that a system conforming to the application role shall receive and process.

A system with such a conformance claim should be tested for each application role specified in the claim. This requires the grouping of testing requirements and test cases for each application role. For each application role, there would be many interactions where the role is either on the receiver or sender side. Therefore, a conformance test case can be designed for each interaction that the application role should support. However, there would be some complex cases where interactions are related (state-transitions). For those situations there would be further testing requirements and test cases.

3.1.1 An Example Use Case Scenario for Conformance Testing - HL7 v3 Document Originator (Application Role is playing the Sender Side)



Document Originator (RCMR_AR000001UV01)

Interactions in which this application role is the sender:

Original Document	RCMR_IN000001UV01
Original Document with Content	RCMR_IN000002UV01
Document Addendum	RCMR_IN000007UV01
Document Addendum with Content	RCMR_IN000008UV01
Document Edit Notification	RCMR_IN000011UV01
Document Edit Notification with Content	RCMR_IN000012UV01
Document Replacement	RCMR_IN000015UV01
Document Replacement with Content	RCMR_IN000016UV01
Document Cancel Notification	RCMR_IN000019UV01
Document Cancel Notification with Content	RCMR_IN000020UV01
Document Repudiation Notification From Originator	RCMR_IN000023UV01

Figure 3.2: An example Application Role

Assume that test cases need to be designed for the application role Document Originator which is an abstract role for healthcare systems that can generate clinical documents. Document Originator role is on the sender side for all its interactions. Figure 3.2 illustrates the HL7 interactions for the application role. By using these interactions, the Document Originator stores, updates, replaces and cancels clinical documents in clinical document management systems.

Assume we first want to test the conformance of the system for Original Document interaction. When a System Under Test (SUT) is playing an application role at the sender side for an interaction, as a first step in testing process, the SUT is requested to initiate the interaction.

In the HL7 case, the trigger event of the interaction can be triggered. When the trigger event of the interaction is user request based, the configuration in the test harness is not difficult. In that case, test designer only should setup a test step which will ask the SUT to initiate the interaction and which will make the test framework to receive the message from the SUT.

HL7 Requirement 1: The Test Framework should have the ability to inform the user of the SUT for the steps in which the SUT needs to initiate an interaction (send a message). In fact, this information should be provided to SUT users before starting the test case in the form of a test case description which can be narrative text or some graphical representation describing test flow and each step. For complex flows it would be better to have notification mechanism (over the user interface) which informs the user when the execution is on the messaging step.

When the message is received, the content should be validated against the constraints defined by HL7 for the corresponding Message Type of the interaction. Since the constraints are represented as XML schemas, test designer should setup a test step which will make XML schema validation of the message according to given XML Schemas.

HL7 Requirement 2: The test framework should enable test designers to setup syntactic validation steps. For HL7 v3, this validation step should do XML validation according to the XML Schemas given by the test designer.

All syntactic constraints defined by HL7 for Message Types are not only covered by XML schemas. For some data element values (coded elements), HL7 allows the usage of external vocabularies like ICD-10, SNOMED CT or LOINC which are collections of coded concepts in medical domain. HL7 defines two types of binding of vocabulary to coded model elements: Static Binding and Dynamic Binding. Static binding means that the allowed values of the value set do not change automatically as new values are added to a value set. That is, the binding is to a single version of a value set. Dynamic binding means that the intent is to have the allowed values for a coded item automatically change (expand or contract) as the value set is maintained over time. For both cases, test designer needs to setup a test step which will go over the whole message and validate all the coded elements according to their corresponding code system. When static binding is used, the valid code list is ready in some format (e.g. simple text file or XML including list of codes) so setting up a mechanism which will check the validity of a given code is easy. On the other hand, for the dynamic binding, maintainers of code lists must provide services which answer the queries for validation of a given code.

HL7 Requirement 3: The test framework should enable test designers to setup validation steps which will check all coded elements in the message in a given code list. When the code list is statically bounded, test designer should be allowed to provide the code list in some format as input. In other case, test designers should be enabled to provide the necessary parameters so that the validation service of the maintainers of code list can be used dynamically in test execution time.

The following interactions for the Document Originator role can similarly be tested. However, in a real life scenario these interactions necessitate an existing document stored in the document repository so that they can make an addendum, edition or replacement on the document. In this case, although the repository part is only a simulation by the testing framework, the Document Originator still needs to know the document reference in order to use it in the messages. In the running example, this problem can be handled by combining all the test steps for each interaction into a single test case. In other words, the SUT playing the Document Originator is first requested to create a document and store it, then make an addendum to the document and then make some edition and so on. Nevertheless, there would be many different cases where the test designer needs to provide some preliminary requirements, knowledge or configuration parameters about the test process to the SUTs before test execution.

Providing preliminary requirements about the message or other testing process to the SUTs becomes more vital when testing the semantic content of documents or messages. Until now in the running example, the SUT is requested to send a message conforming to the Original Document interaction and test steps are setup for syntactic tests. However, by such basic conformance and interoperability scenarios we can only guarantee that the application can send some conformant messages. However, for a real life scenario, it is necessary to guarantee that any message send by the application is a conformant message. This is especially necessary for certification, for better interoperability (both syntactic and semantic) and more importantly to ensure accurate data. In other words we need to test if the information in the generated content that is messages or documents accurately represents the intentional semantics of the users of the application. In the running example, we need to test the capability of the SUT to use a given required piece of information about the clinical document to generate a conformant Original Document message.

HL7 Requirement 4: The Test Framework should enable test designers to express some pre-

liminary information and present this information to SUTs before starting to test execution.

The easiest way to present such information is a narrative text describing the real life scenario in test designers mind about the concerned application roles and the interactions. In this scenario description, the test designer can provide exact values that the SUT should use in the message or some reference or description of information so that users of SUT can easily understand the value that they need to use in the message. The following preliminary description part can be a good example for eHealth domain: "A patient whose name is *John Doe* visits physician *Mary* on *2008-10-13*. The main diagnosis identified after the examination is *Malaria*".

The emphasised parts are possible values of some elements in a message or clinical document. The name of the patient and the physician are of the exact values expected in the messages. The date of the visit is also provided but the date format of the corresponding message element can be anything. When it comes to the diagnosis, it can be the description of the coded value. The standard may necessitate the SUT to set the exact code (e.g. the corresponding ICD-10 code) which corresponds to the given description in the code list.

Presenting the preliminary requirements to the SUTs is the first part of the scenario based testing, in the second part the messages should be tested according to the given preliminary requirements. Therefore, although the preliminary requirements can be presented as narrative text, internally they should be represented in a structured way as a template where the critical information can be changed easily. In this way, the test designer will neither need to update presentation of the scenario nor the test scripts to test the given requirement.

HL7 Requirement 5: The preliminary scenario requirements should be represented in a structured way as a template in order to facilitate the maintenance of test cases.

When a scenario restriction is set on a message element, the test designer also needs to design a test step which will check if the value of the message element is correct according to the given requirement. Some processing mechanism is needed in order to retrieve the value of a specific element in the message.

HL7 Requirement 6: In order to retrieve the value of any attribute or element from the HL7 v3 message, the test framework should support suitable expression languages. For HL7 v3 messages, as they are in XML format, XPATH or XQuery are suitable candidates.

HL7 v3 also defines header like parts, called wrappers, for its messages. These wrappers carry some transport related information and also wrap the actual content. For example, Transmission Wrappers have mandatory attributes which identify the sending and receiving systems of a message. When the test designer wants to test these message elements, it will be not convenient to ask the SUTs to set some configuration parameters, for example device ID, of their system to a specific value. Such parameters and information can be requested from the users of the SUT before starting the test execution.

HL7 Requirement 7: The Test Framework should provide a mechanism to ask the SUTs some parameters and information about their systems before test execution. This mechanism should automatically bind the information into the testcase without any manual intervention so that the corresonding message elements can be tested if they are consistent with provided information.

3.1.2 An Example Use Case Scenario for Conformance Testing - HL7 v3 Clinical Document Directory (Application Role is playing the Receiver Side)

Through this use case scenario, we analyze the conformance test requirement when testing an application role on the receiver side. HL7 Clinical Document Directory is defined in HL7 v3 as follows: "An application role that maintains a directory of completed clinical documents that are housed in the document management systems of contributing organizations."

It is clear from this definition that several medical information systems store their clinical documents and document metadata into this shared system and can query and retrieve documents according to metadata. Assume that a test case is needed for the Clinical Document Directory role for the Find Document Metadata Query interaction. This interaction queries the directory with some metadata (e.g. patient IDs, clinical document type, etc) and the Clinical Document Directory returns the document references as a result. For the conformance testing of this role we need to check the followings:

- The system should accept the valid Find Document Metadata Query interaction
- The system should return all the document references that are intended by the query and this message should conform to the Find Document Metadata Response.

- The system should respond with specified error acknowledgement when the Find Document Metadata Query interaction is not valid

From these test requirements it can be easily concluded that the test designer should be enabled for providing valid and invalid Find Document Metadata Query messages to the test case as input to send to the Clinical Document Directory. However, as it is expected in preliminary requirements, the information reside in the message should be bounded to the test case. For example, if the query is based on patient identifier (request all clinical documents of a specific patient), the test designer needs to use this identifier while checking the response message (Find Document Metadata Response) if it has right patient ID. In such a case, for easy maintenance, the test designer does not need to update the test scripts when he/she wants to change patient ID that is planned to be in the message.

HL7 Requirement 8: The Test Framework should provide a mechanism for test designers to provide the message content that will be sent to SUTs on the receiver side. It would be better if the test designers can provide the content as message template where some information that will reside in the message is bounded to the test case description with some mechanism.

Although the test designer will be an expert on the HL7 standard, creating sample messages for a given interaction,can take considerable amount of time.

HL7 Requirement 9: The Test Framework should be able to integrate tools which can automatically generate sample messages from a given schema.

Even if all these steps are performed, there is a crucial requirement that should be handled before this test case can be initiated. In order to query clinical documents and then check the query results, the SUT playing the Clinical Document Directory should already have some documents. Therefore before these steps, the test designer should setup some test steps to register some documents to the SUT by using Directory Original Document Registration Request interaction. This requires that system need to pass the test case related with another interaction. However the system may not have yet passed the interaction for that step. In fact in this case these interactions cannot be tested individually since there is cause-effect relationship between them.

HL7 Requirement 10: The Test Framework should represent and take care of the relationships between conformance test cases. There are some test cases that cannot be applied to a system *until the system passes another test case.*

Now the designer can setup test steps which will register some documents by the Directory Original Document Registration interaction and then some other test steps which will query the documents by the Find Document Metadata Query. Then some final test steps can check if the correct document references are returned.

How the messages are transported between systems is another issue in testing. HL7 v3 provides a number of transport specification profiles and specifies them as normative. These are ebXML messaging, Minimum Lower Layer Protocol (MLLP) or Web Services. HL7 leaves the decision for the selection of transport protocol to the national or regional affiliates that want to build HL7 based interoperable health network. In any case, when a system claims conformance to HL7 v3, it is expected to select one or more transport protocols that it supports. The test framework, on the other hand, should be capable of testing any of these transport protocols to be able to test HL7 v3 based systems.

HL7 Requirement 11: The test framework should be capable of sending and receiving HL7 v3 messages by using ebXML Messaging, MLLP and Web Services in conformance to the corresponding HL7 Message profiles.

Test designer will also want to test the messages according to the requirements provided in the HL7 message profiles. For example, if the SUT claims to conform to the HL7 Web Service Profile with Addressing Profile option, the testing framework should have the capabilities to let the test designer to develop test steps to check the WS-Addressing headers residing in the SOAP Header of the message.

HL7 Requirement 12: The test framework should enable test designers to also retrieve the parts other than message payload (headers) and to setup validation test steps on these parts.

3.1.3 An Example Interoperability Use Case Scenario for HL7 v3 - "National Health Data Repository" and "Data Consent Story Boards"

This use case scenario addresses the sharing of Electronic Healthcare Records (EHRs) based on HL7 v3 standards. Sharing EHRs is on the agenda of many countries around the world. For example, in the USA, sharing EHRs through National Health Information Network (NHIN)

[35] has become a priority. Similar efforts include the Health Infoway in Canada [36], the NHS Connecting for Health in the United Kingdom [37], the Dossier Medical Personnel (DMP) in France [38] and the National Health Information System (Saglik-Net) in Turkey [39].

The eHealth interoperability use case scenario involves the following HL7 v3 interoperability profiles: The "National Health Data Repository" [40] and the "Data Consent" [41] storyboards with fine-grained constraints imposed by some example healthcare policies.

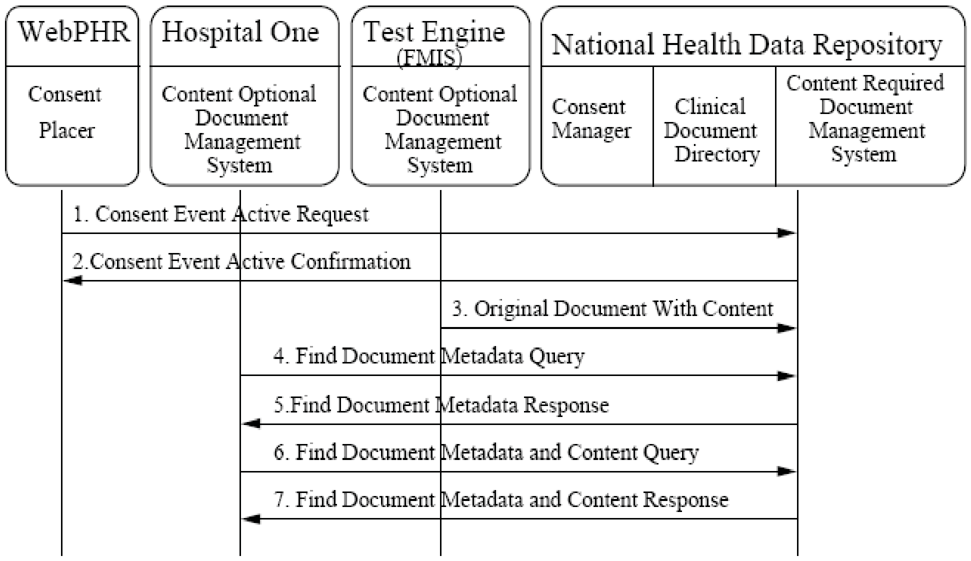

WebPHR
Consent Placer
Hospital One
Content Optional Document Management System
Test Engine (FMIS)
Content Optional Document Management System
National Health Data Repository
Consent Manager
Clinical Document Directory
Content Required Document Management System
1. Consent Event Active Request
2.Consent Event Active Confirmation
3. Original Document With Content
4. Find Document Metadata Query
5.Find Document Metadata Response
6. Find Document Metadata and Content Query
7. Find Document Metadata and Content Response

Figure 3.3: An Example Interoperability Profile

As shown in Figure 3.3, the HL7 v3 roles in this profile are the "Consent Placer" [42] played by a Web Personal Health Portal ("WebPHR") and the "Content Optional Document Management System" [43] played by a healthcare organization ("HospitalOne"). Furthermore, since NHIS National Health Data Repository (NHDR) functions both as a repository for health documents and as a reference point for consent activation and enforcement, three roles have been associated with it: the "Clinical Document Directory" [44], the "Content Required Document Management System" [45] and the "Consent Manager" [46].

There are syntactic and semantic restrictions on the formats and payloads of these transactions. Finally, the profile specifies fine-grained access control rules on the EHR documents. In order to test this use case scenario, the first thing needed is the configuration of the systems so that they can send/receive message to/from each other. For example, the WebPHR system should know the network address of NHDR in order to initiate necessary interactions. Therefore the first step in the testing process should be sharing of configuration parameters of systems.

HL7 Requirement 13: Both interoperability and conformance test scenarios necessitates some configurations on systems. The Test Framework should enable systems to share their configuration parameters among each other.

After configuration, there is a need to inform all the parties about the initial test data. In the given scenario, the initial test data is the patient identifier and the patient demographics information. However, the Test Designer may wish to design a more generic test scenario instead of restricting the tests to some specific patient information. For example, the Test Designer may appoint the administrator of the PHR system to be in charge of determining the patient information. So there is a requirement on how to inform all the related parties about the initial test data.

HL7 Requirement 14: The Test Framework should enable test designers to leave the responsibility of setting the value of some information in preliminary requirements to the users of the SUT. The responsible SUT user should be enabled at runtime to determine the value before the test execution starts.

The example profile requires that the applications playing the HL7 Consent Placer role should be able to produce both fine-grained and record based consents. In order to test if the PHR system has this capability, the administrator of the PHR system is requested to produce a consent document based on the access control descriptions provided by the test writer. An example access control description can be: “Register consent such that mental illnesses of the patient listed in the ProblemList section of the Discharge Summary document are masked and shown solely to the patient’s Psychologist.” In order to test whether PHR system is capable of generating such a consent, it is necessary to intercept this message and check if the consent produced complies first with the profile (syntactically - e.g. the XSD Adaptor) and then with the rule definition given above (semantically - e.g. Schematron validation, XPath evaluation).

HL7 Requirement 15: In interoperability scenario based tests, there is a need to capture and test the message exchanges among the SUTs playing different roles. For this purpose, a proxy *mechanism is needed to act as a mediator which listens to the messages between the systems.*

In the next step, it is necessary to emulate the “Content Optional Document Management System” and the “Original Document with Content” [47] transaction. This step requires the template CDA document provided by the Test Designer to be tailored at run-time according to the semantics of the scenario. There should be mechanisms to use the values such as the patient identifier information initialized by the PHR administrator at run-time.

HL7 Requirement 16: In some interoperability scenarios, test designers may want to setup some test steps to emulate a system in the business process. The Test Framework should provide the necessary capabilities to enable such emulation.

In the next step, the administrator of “HospitalOne” queries NHDR for all “Discharge Summary” documents of the given patient. The “Find Document Metadata Query” [48] and its response messages need to be intercepted for further syntactic and semantic processing. More specifically, the query parameters should be tested to check whether they are set correctly for the document type. This is necessary to validate the conformance of the exchanged documents as well as to check whether the systems involved provide the expected functionalities.

Following the previous interaction, the scenario requires the administrator of “HospitalOne” to authenticate himself to the system as the patient’s “General Practitioner” and issues a “Find Document Metadata and Content Query” [49] to retrieve the desired document. In this test step, it is necessary to have mechanisms to verify that the document is actually the one that has been registered. How NHDR handles the patient consent must also be tested. According to the given access restrictions, the “ProblemList” section in the document should be masked if the access requestor is not a “Psychologist”. Since the role is a “General Practitioner”, the “ProblemList” section in the returned document should be checked to see if it has indeed masked the mental illnesses of the patient. *HL7 Requirement 17 handles this case.*

According to the profile, the “Content Optional Document Management System” application role should properly render the CDA document received. The Test Designer should be able request information from the SUT about the document content to compare the answers with the data contained in the document that it has originally created.

HL7 Requirement 18: The Test Framework should enable test designers to setup test steps which will request any information from SUT users probably by using the user interface. The *gathered information should be able to be used in other test steps.*

3.2 e-Government-Public Procurement - CEN/BIIScenarios

“The purpose of the CEN/ISSS Workshop on Business Interoperability Interfaces for Public Procurement is to provide a basic framework for technical interoperability in pan-European public procurement electronic transactions, expressed as a set of technical specifications compatible with UN/CEFACT in order to ensure global interoperability, using the NES and CODICE customizations of OASIS UBL 2.0 as its starting point.” [50]

Universal Business Language 2.0 (UBL)[51] is a library of XML documents addressing the requirements of electronic procurement and international trade and transportation. Currently, it is in its second version (UBL 2.0) which was released as an OASIS standard in December 2006.

The main aim of NES [52] is to facilitate the establishment of a common platform for e-procurement among its members (Denmark, Norway, UK, Sweden, Finland, Island) and through this platform to facilitate interoperability and practical use of e-procurement in both domestic and cross border trade; to facilitate harmonization of different types of e-procurement documents and to contribute to the development and use of an international standard for e-procurement and it is based on UBL.

CEN/BII workshop is an important tool in the PEPPOL project [53], Pan European Public Procurement project. The Workshop is focused on implementation facilitations and coordinating pilots implementing the technical specifications output. The requirements and final specifications will be input into UN/CEFACT. The starting point for the Workshop is the NES and CODICE customizations of OASIS Universal Business Language 2.0.

CEN BII is producing a set of profiles to define the rules for electronic document exchange between parties when dealing with electronic public procurement across Europe in order to facilitate interoperability. In a real situation, to achieve electronic document exchange among different parties, an agreement has to be settled between the sender and the receiver defining the information being exchanged and the business rules that will govern the exchange of information. This process is usually complex and cannot be automated easily. At this point, the profile concept tries to eliminate the need for bilateral agreement by defining a standard set of choreographies, business rules and electronic business document constraints, to enable users to prepare IT systems in order to be profile compliant, therefore they will be able to exchange documents with all the other users in the community that are also able to handle the same specific profile. CEN BII profiles are technical specifications that describe the followings for a specific use case:

- business processes, i.e. a detailed description of the way trading partners intend to play their respective roles, establish business relations and share responsibilities to interact efficiently with the support of their respective information systems,
- the business rules governing the execution of that business process,
- possible run-time scenarios and the business commitments achieved,
- the electronic messages exchanged as part of the business process and the sequence in which these documents are exchanged,
- the information content of the electronic messages exchanged.

Each CEN BII profile first provides a list of business benefits of the profile and their corresponding beneficiaries. The profile also specifies high level business requirements which set out the scope and main principles of the profile. The detailed description of each profile consists of five main sections. The first section mentions the use cases that the profile addresses. The next section describes the business partners and their roles in the profile.

After defining the roles, the choreography among these roles are described in the next section. As shown in the Figure 3.4, the choreography is defined as the relations among the business collaborations. In addition to these collaboration diagrams which illustrate the sequence of interactions, the profile also provides pre-conditions (e.g. the Customer and Supplier have identified each other), business rules on choreography level (e.g. If documents make references to contracts such as framework agreements those contracts supersede the document content.), description of each step and post-conditions (e.g. Invoice has been received).

In order to help software vendors and implementers to integrate CEN BII profiles into their systems, CEN BII decides to produce a abstract Validation Architecture report which describes how some validation artifacts (e.g. XML Schema and Schematrons) should be derived

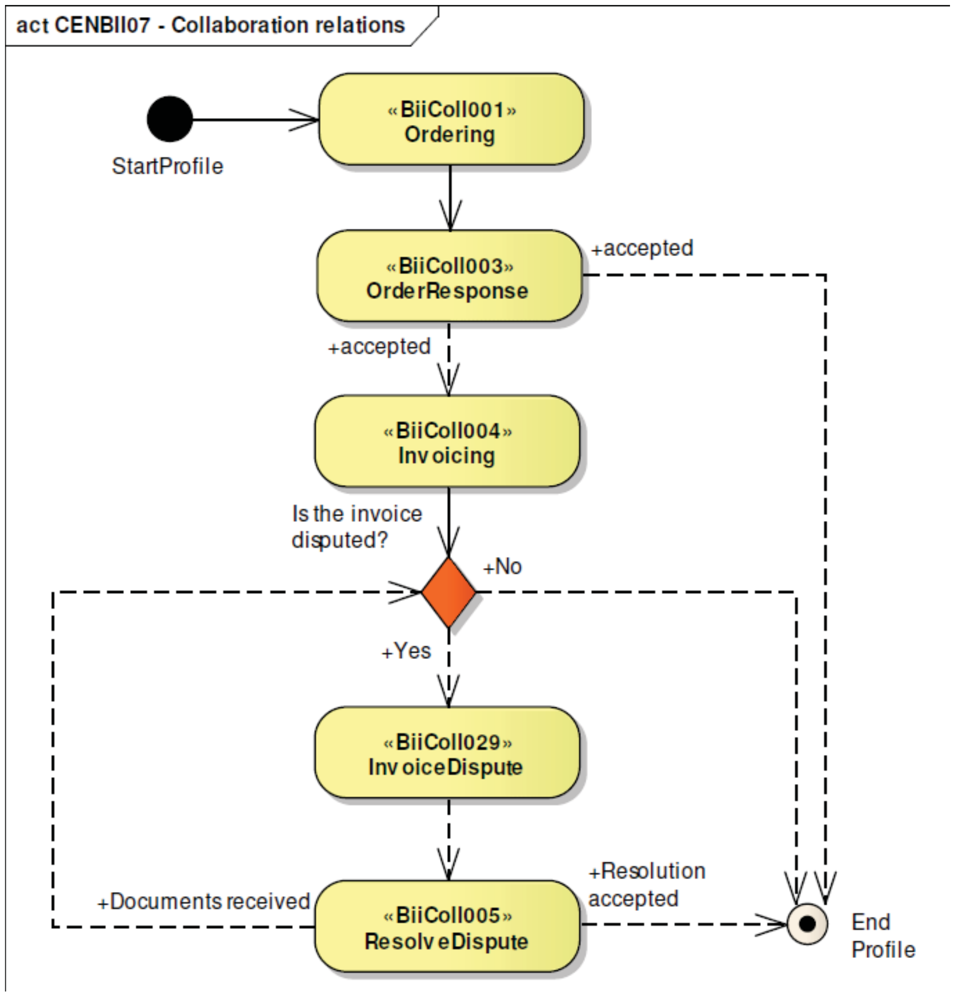


Figure 3.4: CEN BII Simple Procurement Profile - Choreography

from the business rules and specifications mentioned in the CEN BII profiles. In their Conformance and Interoperability Testing Validation Architecture Report, they also identify the following generic requirements for the validation architecture:

- Every electronic business document instance MUST be well formed.
- Every electronic business document instance MUST be valid according to its related grammar-based language definitions.
- Every electronic business document instance MUST be valid according to the rules defined in the profile.
- Documents exchanged in the context of a CEN BII profile SHOULD follow the defined

choreography.

- CEN BII validation architecture MUST be syntax neutral.
- Validation artifacts MUST be easy to maintain and update.
- Validation artifacts MUST work on different platforms.

3.2.1 An Example Use Case Scenario for Conformance Testing - CEN BII Simple Procurement Profile (Application Role is playing the Customer side)

In this section, an example conformance test case for the Customer role of CEN BII Simple Procurement Profile will be illustrated. The collaboration diagram of the profile is shown in Figure 3.4. As shown in the diagram the profile describes the business document exchanges starting with the order of some items from a supplier, the response to this order and if accepted the transmission of the invoice. The second part continues with the application response to the invoice and some further business documents depending on some decisions. The business flow includes decision points where the flow will follow one of the branches according to the decision of customer or suppliers. From the branching options, the following scenarios can be derived:

- Accepted Order, Accepted Invoice: When the seller accepted the Order and the buyer accepted the Invoice.
- Rejected Order: When the seller rejected the Order.
- Accepted Order, Invoice Overcharge: When the seller accepted the Order but the buyer then rejects the Invoice because its value exceeds that of the Order. The scenario continues with the Credit transaction sent by the seller.
- Accepted Order, Invoice Undercharge: When the supplier accepted the Order but the customer then rejects the Invoice because it is less than that of the Order. The scenario continues with an extra Invoice sent by the seller.
- Accepted Order, Replace Invoice: If the Creditor decides to “correct” the disputed invoice by replacing it, the Creditor creates and sends a Credit transaction and sends a new replacement Invoice transaction the Debtor.

In order to claim that an application conforms to the CEN BII Simple Procurement Profile, the application should be successful in all of these scenarios. Therefore, we need at least one test case for each scenario for conformance testing of this profile. Assume that a test case will be designed for the Customer side for the CEN BII Simple Procurement Profile - Accepted Order, Invoice Overcharge scenario. Consequently, the test designer will setup necessary test steps to simulate the Supplier side.

Before starting the test case, the user of the SUT should understand the flow of the scenario so that he/she can operate the system correctly. Hence the scenario flow, the order of business document transitions, the branching points (e.g. Accept/Reject Order) in the scenario should be described clearly to the SUT user. Furthermore, the user should be able to monitor the execution of the flow during the test case execution.

CEN BII Requirement 1: The test framework should have the ability to present the scenario flow to the SUT user in a descriptive way. This presentation can be done by narrative text however it would be better if it can also be presented as flow diagrams similar to the diagrams given in CEN BII Profiles.

CEN BII Requirement 2: The test framework should enable the users of the SUT to monitor the test execution flow during the test execution. In this way, they can also timely respond to the instructions that they should perform.

Having these abilities, when the test designer setups a test step to handle the Ordering collaboration that is to receive the Order document from the buyer, the user of the SUT will be notified by the instruction of the step. When the Order document is sent, the conformance of the document to the structure defined for the CEN BII Submit Order transaction and the business rules defined in the profile should be checked. In order to perform these conformance checks, the test designer should setup several validation test steps. First validation should be against the XML schema provided by CEN BII for the Order document.

CEN BII Requirement 3: The test framework should enable test designers to setup syntactic validation steps. For CEN BII Business documents, this validation step should do XML validation according to the XML Schemas provided by CEN BII.

Validation against the XML schema is not enough. As in all electronic business messaging standards, code lists like currency codes and country codes play an important role in CEN BII profiles and the base UBL standard. UBL 2.0 recommends a two step validation model since the specification of the default values directly in the schemas makes it difficult to modify the code lists to meet customization requirements. In the first phase of the two-phase validation, an incoming invoice document is validated against UBL 2.0 XSD schemas. If the instance passes the first phase, in the second phase it is checked against the business rules. These rules are specified through Schematron language in CEN BII abstract validation architecture as it is done in NES. If the instance passes both of the phases successfully, it is delivered to the processing business application.

CEN BII profiles identifies its own code lists. UBL recommends creating a XSLT file from the code list configuration files to use it in validation. They also define a methodology, UBL Code List Value Validation Methodology, for this process.

CEN BII Requirement 4: The test framework should enable test designers to setup validation steps which will check all coded elements in the business document. For NES and UBL, this validation will be done by XSLT file provided by the test designer. XSLT can be designed with the UBL Code List Value Validation Methodology according to the code lists specified by the NES. The output of the XSLT should be used as test step report.

Creating an XSLT file for all code lists can be difficult for test designers. Instead, they can create code list configuration files where each of them is a XML file in a specific format listing all codes for a code list. In that case, a special validation component is needed to validate the coded values in a business document according to code list provided by test designer in code list configuration file format.

CEN BII Requirement 5: The test framework should allow integration of different validation components as plug-in so that test designers can select the most suitable validation methodology for specific test steps according to the auxiliary testing materials (e.g. XSLT files, UBL code list configuration files) they have.

The abstract validation architecture of CEN BII describes another alternative for code list validation. In this alternative, they use schematron files for code list validations.

Until now, the test steps for schema validation for the Order document and code validations according to the code lists provided by CEN BII have been designed. Nevertheless, as already mentioned CEN BII defines rules on several levels (profile level, collaboration level, and transaction level) on business documents. Instead of providing a refined XML schema for each profile, CEN BII defines the refinements as business rules. The business rules are provided with Schematron files. Therefore, the test designer should setup a validation test step to validate the Order document according to the corresponding Schematron. In our scenario, for the Submit Order transaction the following rules are identified: i) Quantity on line must be positive, ii) Amounts must be positive.

CEN BII Requirement 6: The test framework should enable the test designer to setup a validation step which requires a Schematron as input and will do Schematron validation when executed. The output of the Schematron should be used as test step report.

The test scenario presented does not specify about the details of the scenario except from the message flow. For example, items the customer will order, amount of items, party information for supplier or tax details are not specified. Therefore, the SUT is expected to generate an Order document with arbitrary information. However, this type of test cases is not recommended as they do not completely test the conformance and interoperability of applications. Like the applications we face in real life, we need to test their ability to create a conformant business document when the information about the document is specified. The CEN BII Order document has many optional parts but for the required elements, the test designer needs to specify some information as the scenario requirement. For optional elements, further test cases may be designed.

CEN BII Requirement 7: The test framework should enable test designers to provide the scenario requirements (the information for business document contents) which will be presented to SUTs so that they can operate accordingly.

Once the user of the SUT gets the required information, he will supply it to the application to generate the expected Order document. The test designer needs to setup some test steps to check these requirements. XPATH expressions or Schematrons can be used for these test steps. The expressions will be simple value comparison expressions that compare the information given as the requirement and the corresponding data element value in the business document. To facilitate the maintenance of these expressions and the test case, the requirements should be represented as variants in the expressions. In this way, test designers can easily change the requirements without modifying the expressions.

CEN BII Requirement 8: The test framework should enable test designers to setup test steps to realize value comparison for data elements. The expressions for these comparisons should bind the value of scenario requirements to some kind of representation in order to facilitate the test case maintenance.

In the next phase, after testing the Order document syntactically and validating it against the scenario requirements, the test designer needs to setup a test step which will create and send an Order Response Simple document to the customer that is handling the OrderResponse collaboration with the AcceptOrder transaction. The test designer needs to supply the content of this document in design time. However, many parts of the document need to be determined in run time (test case execution time). For example, the collaboration level rule "OrderReference must be present" and transaction level rule for AcceptOrder transaction "An AcceptOrder must provide a reference to the Order for which it is a response" necessitates the run-time assignment of the value of the OrderReference in the Order Response Simple document. In other words, it should be copied from the Order document that the SUT sends in previous step. Another example is the content of the BuyerCustomerParty element in the Order document. It provides general information about customer firm (company name, address, contact, etc) and should directly be copied to Order Response Simple document. More examples can be given for these cases.

Moreover, although the test designer may determine some scenario requirements and these requirements can be the basis for the Order Response Simple document in design time, he can defer some decision or content generation to run time. For example, the test designer may not specify the number of items that will be ordered in the scenario. In that case, some data elements like LegalMonetaryTotal (the total value of the invoice), TaxTotal, InvoicedQuantity in the Invoice directly depends on the number of items specified in the Order document. The test designer should be able to setup some test steps to change the content at run-time.

CEN BII Requirement 9: For business documents that will be sent to the SUTs, the test framework should enable test designers to provide the document content in design time. In accordance with the test scenario, this content can be real content that will be directly used or a content template that will be updated during the test execution by further test steps.

Dynamic content construction from an XML document template requires some abilities such as setting the value of an element or attribute and adding an XML fragment to a specified location in the document. The test framework should also allow test steps to do calculations (in order to calculate total monetary value or tax amounts) or other simple operations on strings.

CEN BII Requirement 10: The test framework should enable test designers to setup test steps to do special data processing and create new data. Inserting an XML fragment to a specified location in a document, setting the values of elements and attributes in an XML template or making some arithmetic calculations are examples for such processing instructions.

In summary, as a test designer until now we specify some scenario requirement, setup test steps to receive the Order from customer, validate the Order document against schemas, code lists, business rules and scenario requirements, setup test steps to generate and send the Order Response Simple document and setup test steps to generate and send the Invoice document. However, before the test steps regarding the Invoice there is actually one business step that should be performed. In real life, it is the delivery of ordered items from supplier to customer. In a test scenario, it is actually an imaginary delivery by informing the SUT that delivery is assumed to be completed. In addition some information about delivery should also be provided, since the overcharge situation in the CEN BII Order Accepted, Invoice Overcharge scenario can be related with delivery. For example, if the number of delivered items is less than the number of items accounted in the Invoice, the situation is assumed to be Invoice Overcharge.

CEN BII Requirement 11: The test framework should enable test designers to setup intermediate test steps which will interact with the SUT user over the graphical interface and provide some information about the running scenario.

Another issue to be addressed is the transportation protocols that are used to transport business documents between customer and supplier. CEN BII and UBL are providing specifications on business documents and do not specify the communication or the transport layers normatively. Therefore, if the objective is designing generic test cases which test only the conformance to CEN BII Profiles, the transformation of business documents can be via graphic interface with document upload and download capabilities. In other words, when a SUT needs to send a document according to the scenario, he needs to capture the document that his application produces and upload it from the graphic interface. Similarly response documents from the test framework side can be made available for download.

CEN BII Requirement 12: When a specific communication or transport protocol is not specified in a profile or standard, the document exchanges should be realized over graphic interfaces. The test framework should enable test designers to setup test steps which will interact with the SUT user and the test framework will get the business document over graphic interface uploaded by the SUT or provide a document created in the scenario for SUT to download.

However, any real life application will be in need of some communication and transport protocols. For example, in PEPPOL the transport layer will be based on Web services. When a specific communication protocol is selected for the implementations, it should be possible to easily update the test cases to support the selected protocol.

CEN BII Requirement 13: When a specific communication or transport protocol is specified, the test framework should enable test designers to setup test steps which have the capability to send or receive business documents based on the selected protocol.

For the remaining business transactions, similar test steps can be designed.

3.2.2 An Example Interoperability Use Case Scenario for CEN BII - Simple Procurement Profile

In this scenario, the interoperability of applications is tested against Simple Procurement Profile. The only difference from the conformance test scenario is that this time there are two applications under test one playing the Supplier role and the other playing the Customer role. Assume that an interoperability test case will be designed for the overall scenario to test the interoperability of two applications which are claimed to be conformant to Simple Procurement Profile. For such an interoperability scenario, the first step should be the configuration management to assure that both parties configure their systems to send or receive business documents to or from each other.

CEN BII Requirement 14: The test framework should also incorporate the automation of the configuration management into the test case execution for both conformance and interoperability scenarios.

Then we can continue with the design of the main test steps. The supplier will send the Order to the customer as first step. In order to test this business document we should be able to capture it. For this test case, we can assume that a specific transport protocol is agreed among parties.

CEN BII Requirement 15: In interoperability scenarios, there is a need to capture and test the message exchanges among the SUTs. For this purpose, the test framework should enable test designers to setup special test steps which will listen to the communication among SUTs and retrieve the content.

Assume that for this scenario, we also want to check if the system is interoperable with the Invoice Overcharge scenario. Therefore, after the test step that will listen the Order Response Simple document exchange, we need to inform the SUT users about the imaginary delivery and provide the appropriate information so that the situation is either overcharge or undercharge. This requires some calculations and extraction of some values from exchanged business documents (e.g. the amount of items ordered given in Order document). After that, the scenario continues with the SubmitInvoice, DisputeInvoice and CorrectWithCredit transactions. However, the next step is conditional and depends on whether the Invoice and Credit Note sent by the Supplier totally matches with the Order. As the test designer we cannot interfere in the decision since the Supplier can make the calculations wrong for Credit Note and this is not about CEN BII conformance. Therefore, the test steps should cover all branches in the scenario. The test designer should setup the test steps that will check the Credit Note and the Invoice and will decide whether they match the Order. Then the test case flow should be branched according to this decision. If a match occurs, the scenario will end otherwise scenario will continue with the Application Response and other documents depending on the situation becomes overcharge or undercharge.

CEN BII Requirement 16: The test framework should enable test designers to setup some branching (decision- if then else) test steps. The decision will be made by some conditional expression and the test case flow will continue with a branch according the result of the expression.

If we analyze the scenario, we can easily understand that this overcharge and undercharge situation can go forever until the total value of sent Invoices and Credit Notes match with the Order. In order to handle such situations, test designer needs some loop mechanism.

CEN BII Requirement 17: The test framework should enable test designers to setup some *special test steps to repeat a set of test steps until a condition holds.*

CHAPTER 4

THE TESTBATN FRAMEWORK

TestBATN framework provides a design and execution environment for dynamic, configurable and automated execution of conformance and interoperability testing against B2B standards, profiles and specifications.

In summary, the TestBATN framework has of the following features:

- An XML based computer interpretable test language (TestBATN TDL) allowing dynamic set up and automated execution of test cases.
- A Test Execution Model defining the meaning of the behavioral and operational semantics of each TDL instruction in an unambiguous way.
- A set of interfaces (TestBATN Module Interfaces) which enables development and integration of pluggable modules supporting different protocols, formats or methodologies and to make these modules available in execution of the specific TestBATN TDL instructions.
- A Web-based Test Control and Monitoring GUI enabling users to monitor and drive the execution of test cases.
- A Test Design and Framework Management GUI enabling responsible organizations to develop, deploy and maintain test cases, supplementary materials for any B2B standard, specification or profile

With these features, TestBATN framework can be utilized as;

- a conformance test platform where vendors can test their systems' conformity remotely over the Web,
- a interoperability test platform where vendors can test the interoperability of their products against each other remotely over the Web, or in a testing event (e.g. ETSI Plugtests, IHE Connectathons), or
- a means of testing for the certification of products against a B2B standard or profile.

4.1 Main Concepts and Testing Approach

TestBATN framework allows integrated testing covering all layers of interoperability stack; the business process layer, the document layer and the communication layer. Rather than testing each layer independently with specific tools and integrating the results, TestBATN has an integrated testing approach which permits test restrictions on all of these three layers of the interoperability stack within the scope of a test scenario.

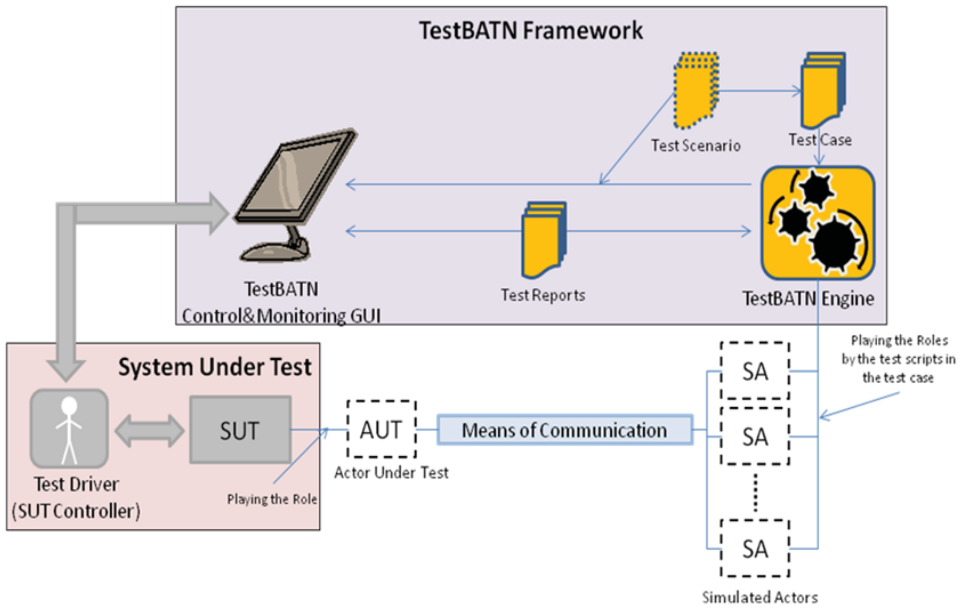


Figure 4.1: TestBATN Conformance Testing Setup

The Figure 4.1 illustrates the main concepts of the framework and the setup for conformance testing. A *Test Scenario* is the abstraction of the complete description involving the participating entities and the steps required to achieve a specific test purpose; testing the systems against the conformance or interoperability requirements of (a part or the whole of) a standard or a profile. The TestBATN framework follows the business process terminology and calls the participating systems of the test scenarios as parties.

Generally, *Test Parties* correspond to the abstraction of the framework for the actors specified in the business process in the base standard. In the TestBATN framework, Test Parties can be of two types; the Simulated Actors or Actors Under Test (AUT). The parties that are specified as AUT are the entry points for Systems Under Test (SUTs) to participate in the test execution. The simulated parties are the actors that take part in the business process or message choreography specified in the base specification, but that are not intended to be tested in the test scenario. They are simulated by the framework by means of test scripts to achieve the test purpose.

The *Test Steps* mentioned in the Test Scenario description are the abstraction for actions or instructions that will be performed by the Test Framework or by the SUT to execute the test case. The backbone of these steps is the message choreography specified for the Test Scenario (or derived from the base specification conformance or interoperability criteria). Other steps are the instructions to the Human Test Drivers controlling the SUTs and the validation steps that will contribute to the test verdict.

As already mentioned, Test Scenario is the abstraction and it should be interpreted as the sequence or flow of actions in Test Designer's mind to achieve the test purpose. In order to automate the test execution, this description is defined in TestBATN machine readable language; TestBATN TDL. By following the conformance and interoperability testing literature terminology, this definition which describes a test scenario is called Test Case.

4.1.1 Conformance Testing in TestBATN

The Figure 4.1 shows the basic concepts and setup for conformance testing. During the test execution, two entities; TestBATN Engine and TestBATN Control and Monitoring GUI take part at the framework side. The TestBATN Engine is the system which interprets and executes the Test Case definition and manages the whole process. The TestBATN Control and Monitoring GUI is the interface between the engine and the Human Test Driver to provide real time monitoring on the test execution, some control on the scenario and reporting of the test results. The Human Test Driver, while monitoring the test execution with the “TestBATN Control and Monitoring GUI”, controls the SUT according to the instructions shown through the GUI.

The purpose of conformance testing is to determine conformance of a single implementation against a particular standard. Therefore, in the setup shown in Figure 4.1, there is only one SUT playing the role of the Actor Under Test (AUT) specified in the conformance test case. The TestBATN Engine, on the other hand, can simulate more than one party according to the business process specified in the base standard or according to the testing purpose of the test scenario. The Means of Communication is the physical setting that enables communication among systems which can be Internet, local network or any other instrument.

For any given testing instance, there will be a test report which includes the final test verdict, intermediate test verdicts for each step, and the detailed reports showing the logs and errors for each step. In fact, the reports and the verdicts for each step are shown to the Human Test Driver at run time without waiting the end of the test case. The interface between the TestBATN engine and the TestBATN Control and Monitoring GUI realize this run-time monitoring facility. In addition, a presentation model is defined in the framework and used to present the test scenario to the Human Test Drivers in a more user friendly way.

4.1.2 Interoperability Testing in TestBATN

The Figure 4.2 illustrates the interoperability testing setup of the TestBATN when there are two SUTs. The situation is similar when there are more than two SUTs. There is not much difference between the conformance and interoperability testing in the testing process. Since the purpose of interoperability testing is to check the end-to-end functionality between two or more communicating systems, usually there is no simulated actor. However, interoperability test scenarios including simulated actors can be designed for more complex tests. Generally, in the interoperability test scenarios, the aim is to listen to the messaging between SUTs. The TestBATN framework achieves this by its specific instructions for listening. The monitoring and reporting process is the same for conformance testing setup.

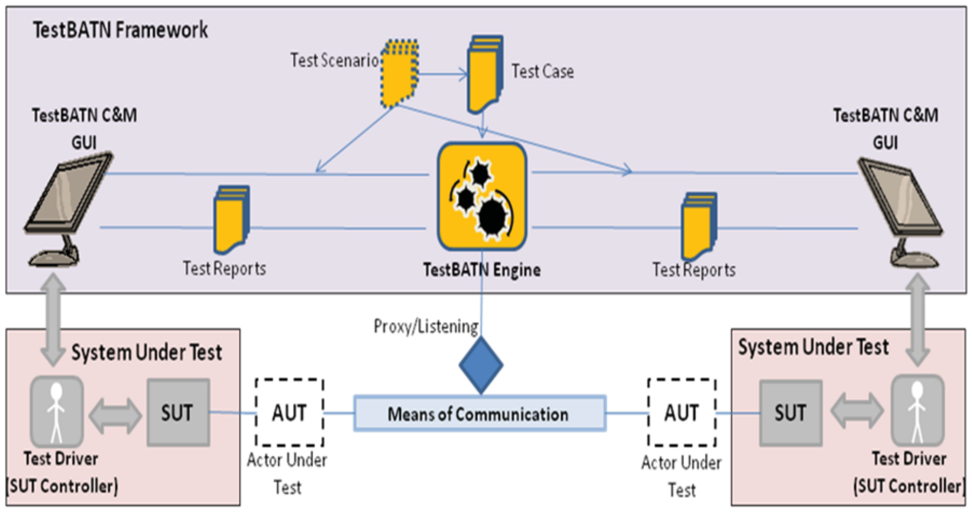


Figure 4.2: TestBATN Interoperability Testing Setup

4.1.3 Scenario Based Testing

As already mentioned the TestBATN framework provides integrated testing based on a scenario for both conformance and interoperability testing and the backbone for this scenario-based testing is the identified actors and the messaging choreography among them. On this scenario skeleton, test designers can set further test steps to check conformance or interoperability requirements of the base standard regarding the business documents, details on business processes, and the communication protocols. However, more information on the business collaboration will be needed in order to execute the test scenario. Current approaches provide this information in the narrative test case descriptions before the test case executions. Then the test cases are configured at design time accordingly to check these further scenario requirements. TestBATN framework develops an innovative approach to scenario based testing and provides mechanisms to set and check these scenario restrictions in the test case definitions at run time.

In the scenario based testing approach, a real life scenario regarding the backbone business process is presented to the Human Test Drivers and they are expected to operate the SUTs according to this scenario. For instance, assume it is required to test a standard which defines rules for the messaging interaction between healthcare institutions for sharing an EHR record. The following text can be a part of the scenario requirements:

- *A patient whose name is John Doe visits doctor Mary in 2008-10-13. The main diagnosis identified after the examination is Liver Cancer*

In this scenario fragment, the underlined words should be the values of the elements in the message if this scenario happens in real life. Since the base specification and standard specify where these values should be located in the message, for real conformance and interoperability, the application should have the ability to use the information given in real life and generate a message or resulting document correctly with this information. Therefore, in addition to syntactic testing, the semantics of the message should also be tested in the scenario based testing.

Scenario based testing is useful but not effective much when all the scenario requirements are provided at design time where the test scripts are configured according to these requirements. In order to make the test scenarios dynamic, configurable and easy to maintain, the TestBATN framework provides a mechanism to present the scenario in structured way like a template. In this way, designation of actual scenario requirements can be delayed until run-time. In TestBATN framework, this phase is integrated in automated test case execution as the preliminary phase. When the requirements are specified during the test execution time, the test scripts in the test case definition should also be configured automatically. The TestBATN framework provides following three alternative mechanisms for each step while developing a scenario template:

- An element in the template can be specified at design time by the test designer
- An element in the template can be specified by one of the Human Test Drivers at run-time
- An element in the template can be specified by a special application which can generate randomized values

The Scenario Based Testing approach of the TestBATN framework facilitates the development and execution of test cases with requirements from real life. In this way, it not only guarantee that the application can send some conformant messages, but also guarantee that in any real life scenario the application sends the conformant message. Scenario based testing approach is more important for certification, for better interoperability (semantic interoperability) and to ensure accurate data exchange. In other words, by using this approach, it is possible to test if the information in the generated content, that is the messages or the documents, accurately represents the intentional semantics of the users of the application.

4.1.4 Modular Approach

The TestBATN framework is a generic testing framework supporting any B2B standard, specification or protocol for conformance and interoperability tests. Current B2B standards specify a variety of communication protocols, content formats or choreographies. In order to support all of these and perform automated conformance or interoperability tests against them, the TestBATN framework is designed as an adaptable and modular software framework. The aim is to automate the manual tasks required for testing process by the use of adapters coded in a programming language. Such adaptors can be developed more easily to perform those jobs. Utilizing specialized adapters to enhance the automated testing process is not novel and followed by other test beds or approaches (e.g. ETSI TTCN-3). However, in these approaches, testing adapters are prepared for specific testing events or specific test cases without a possibility to reuse. The approach of TestBATN framework is defining a common interface for each specific functionality that need to support different protocols, formats, or technologies in testing process or which may facilitate reusability among different test cases from different B2B standards or profiles. Hence as long as the adaptor is wrapped to conform to the interface, any adaptor can be plugged into the test framework.

The TestBATN framework defines the following interfaces:

- Messaging Interface

  Transport-Communication Adaptor Interface: This interface facilitates plugging in adaptors to receive, send or listen to messages by different protocols like TCP, HTTP, SMTP, etc.

  Packaging Adaptor Interface: This interface facilitates pluggable adaptors which will be used to pack or unpack messages according to higher layer messaging protocols such as SOAP or ebMS.

- Test Adaptor Interface

Validation Adaptor Interface: This interface facilitates pluggable adaptors which will be used to validate a content according a schema and which to generate a verdict and a structured test report for the performed validation. The example validation adaptors include XML Schema Validator or Schematron Validator.

Verification Adaptor Interface: This interface facilitates plugging in adaptors to perform complex tests on any content and to generate a verdict and a structured test report for the performed tests. The example verification adaptors include XPATH Verifier, Regular Expression Verifier, other special purpose test tools.

- Value Initiator Adaptor Interface: This interface facilitates plugging in adaptors to generate or to chose random values from a specific value list. The values are used at run-time to generate a message content or a scenario requirement.

- Function Library Interface: This interface facilitates definition and implementation of functions to be used in TestBATN TDL expressions as auxiliary data processing entities. These functions are similar to XPATH functions used in XPATH expressions.

- Data Model Interface: This is the interface to define specific data models for the content formats that are not XML based. TestBATN framework internally transforms them into Object model and enables expressions and other TDL instruction to run over this data.

The Figure 4.3 illustrates the approach of TestBATN framework to enhance reusability for test components or tools. According to the needs of a standard or a domain, adaptors are developed implementing the corresponding interfaces defined in the framework. The developers also provide descriptors for their adaptors explaining the adaptor's main functionality, its input, output and configuration parameters in a structured XML format. The XML schema for these descriptors are given in the Appendix A.2, A.3, A.4, and A.5. When the descriptor and the implementation for the adaptor are deployed into the framework, the adaptor is ready to be used in test case definition and execution.

The Figure 4.4 shows how the Test Designer plugs in a specific adaptor for a TestBATN Test Description Language (TDL) instruction. By considering the test scenario, he chooses the suitable adaptor from the registry of adaptors of a specific type and mentions the ID of the adaptor in the instruction to inform the TestBATN engine to use the adaptor while executing

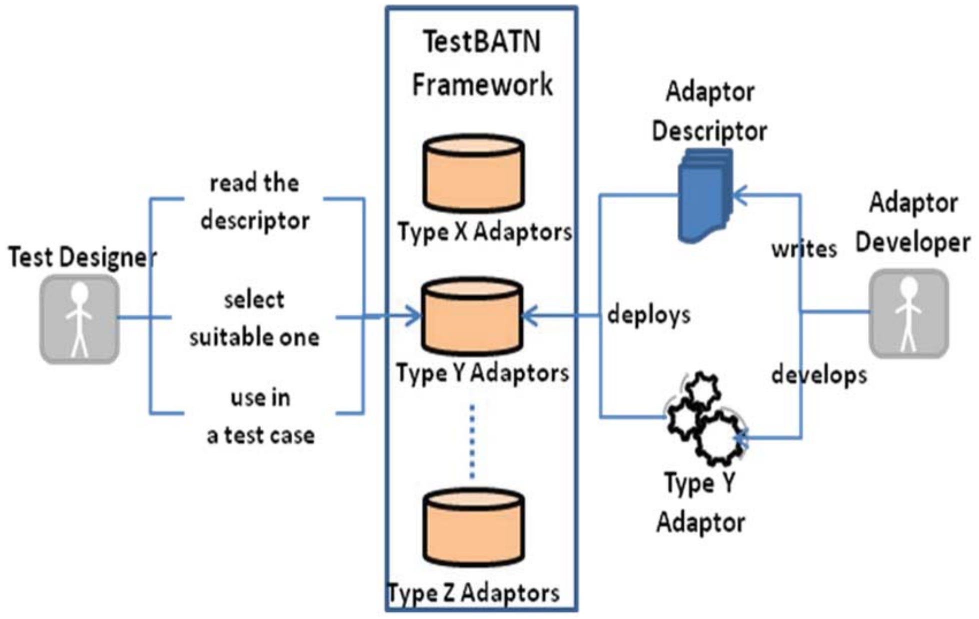


Figure 4.3: Reusability of Adapters

the instruction. Then he provides values for input, output and configuration parameters for the adaptor. Appendix B provides sample descriptors for some default TestBATN messaging and validation adapters.

4.2 Execution Model and Test Description Language

In this section, the constructs (instructions) of TestBATN TDL and their operational semantics are described. The complete XML schema of TDL is given in the Appendix A.1. For any instance of conformance or interoperability testing process, there are the following six phases:

- Test Initialization: In this initial phase, the Human Test Driver selects the test case to be executed and initiates the execution by using the "TestBATN Control and Monitoring GUI". For conformance test scenarios, since there is only one Actor Under Test (AUT), the Human Test Driver selects this actor and start the execution. Then the execution continues with the next phase. For interoperability scenarios, more than one SUT participates to the instance of testing by taking the responsibility of different AUTs. The Human Test Driver for the first SUT, after selecting an AUT, starts the execution and waits for the other SUTs to participate to the test instance. When all the related AUTs

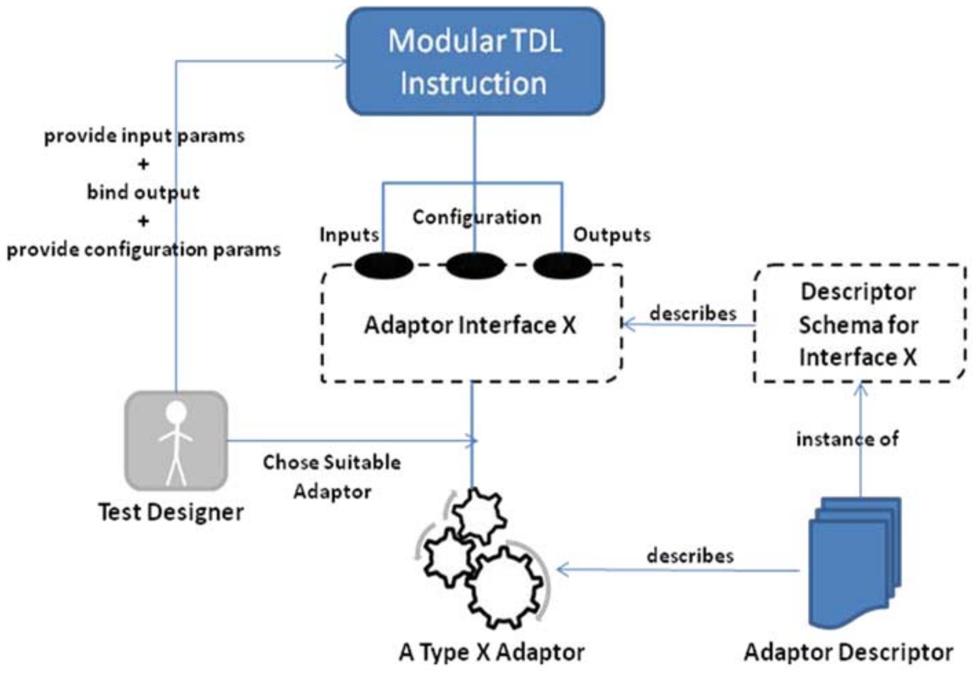


Figure 4.4: Plug-in an Adaptor to a Modular TDL Instruction

join the testing process, the metadata and the description of the test case are presented to the Human Test Drivers and the execution continues with the next phase.

- Configuration Management Phase: In this phase, configuration parameters (network configuration; IP addresses, port numbers, etc) of each SUT and the configuration of the test engine for Simulated Actors are shared among test participants by using the "TestBATN Control and Monitoring GUI". Each Human Test Driver should provide the configuration information of its SUT with the GUI. Then each of them should configure their SUTs with the other SUTs' or test engine's configuration information. The TestBATN engine also configures itself with the information it gathers from the SUTs.

- Scenario Establishment Phase: As described in Section 1.2.3, in this phase, the scenario requirements are shown to the Human Test Drivers. Human Test Drivers determine the scenario requirements that they are responsible (fill in the scenario template) and the finalized requirements are shown to all Human Test Drivers.

- Test Execution Phase: When all Human Test Drivers are ready for execution phase, the test steps are executed in the sequence as described in the test case definition and Human Test Drivers are expected to operate SUTs according to the test scenario.

- Reporting Phase: In the TestBATN framework, the Human Test Drivers can examine the test reports (the verdict and the detailed report) for each test step from "TestBATN Control and Monitoring GUI" after the execution for the step ends. When the execution of the test case finishes, an overall verdict is calculated from the test step verdicts and shown to the Human Test Drivers.

4.2.1 Basic Test Constructs

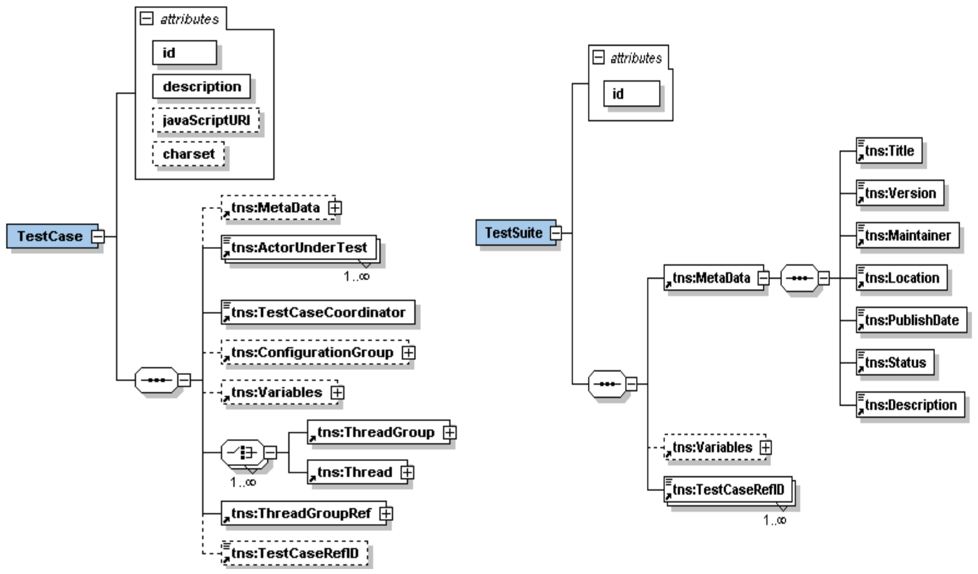


Figure 4.5: Data Model for TestSuite and TestCase Definitions

For a standard or a profile, there can be many test case definitions which target a different part of the specification. In the TestBATN framework, the well known TestSuite concept is used for grouping test case definitions. For example, in testing the "NES Simple Procurement Profile", there are different branching options in the profile depending on the acceptance or rejection of the "Invoice". A test case can be developed for the acceptance case and another for the rejection case. Since the SUTs must behave correctly under both test cases; these test cases can be grouped as a test suite implying that a SUT must be successful in this TestSuite to claim conformance to the "NES Simple Procurement Profile". Test Designers must provide a *TestSuite* definition (a separate XML file describing a Test Suite; a group of test cases) for each Test Suite. The data model for TestSuite definition is illustrated in Figure 4.5.

Basically, a *TestSuite* definition consists of a) an id attribute which is the identifier in the framework for the test suite, b) Metadata which describes some basic information; title, description, date, status of the test suite, c) Variables which is a list of parameter definitions to declare or store global data, and d) TestCaseRefID which is the list of identifiers for the test case definitions in the test suite.

As shown in Figure 4.5, the root element for test case definition is the *TestCase* element. A *TestCase* definition includes the descriptions for all steps that are needed for the automated execution of a test scenario (all six execution phases). Each *TestCase* is identified by a unique identifier (id attribute) in the framework. The Test Designer should also provide a brief description for the test case (description attribute). This description is used while listing the test cases to Human Test Drivers of the SUTs. For a complete description of the test case, the Metadata element can be used. The Test Designer can describe the scope, the base scenario and the testing methodology by narrative text in the Metadata element. The content of the Metadata element is shown to the Human Test Drivers in the Test Initialization Phase.

As mentioned before, each test case is based on the abstract "Test Parties" concept. The *ActorUnderTest* and the *TestCaseCoordinator* elements provide the name of these parties; AUTs and Simulated Actors respectively.

Just like coding in a programming language, while defining test cases, the Test Designer will need variables to store or retrieve information for the consequent test instructions. *Variable* elements are used for storage and retrieval of all types of information like the messages received from SUTs, the specific parts of these messages, or the information coming from Human Test Drivers over the GUI. The scope of the Variable definitions is the whole test case. The TestBATN framework defines default data types like number, string, or XML for the data used in the test instructions. The framework also allows definition and deployment of user-defined Object Data Types. The type of the variable should be either one of the default data types or the user-defined data types. The Test Designer can also assign an initial value for the variable.

The TestBATN TDL facilitates the use of scripting functions in the expressions. These functions can be defined in the pluggable Function Libraries or through Java Scripts where the function definition is deployed to the framework together with the test case definition. In this way, the Test Designers can code some re-usable operations with java script as function definitions and call the functions from the expressions in the test instructions. The Test Designer should provide the relative path of the java script file in the test case package with the javaScriptURI attribute.

The *ConfigurationGroup* element includes the instructions for the preliminary phases: Configuration Management and Scenario Establishement. The actual test steps for Test Execution Phase are defined in *Thread* and *ThreadGroup* constructs. They are the basic constructs for sequencing, concurrency and flow among test steps. The *ThreadGroupRef* element which refers to a ThreadGroup definition in the test case gives the entry point for execution.

4.2.2 Configuration Phase and Instructions

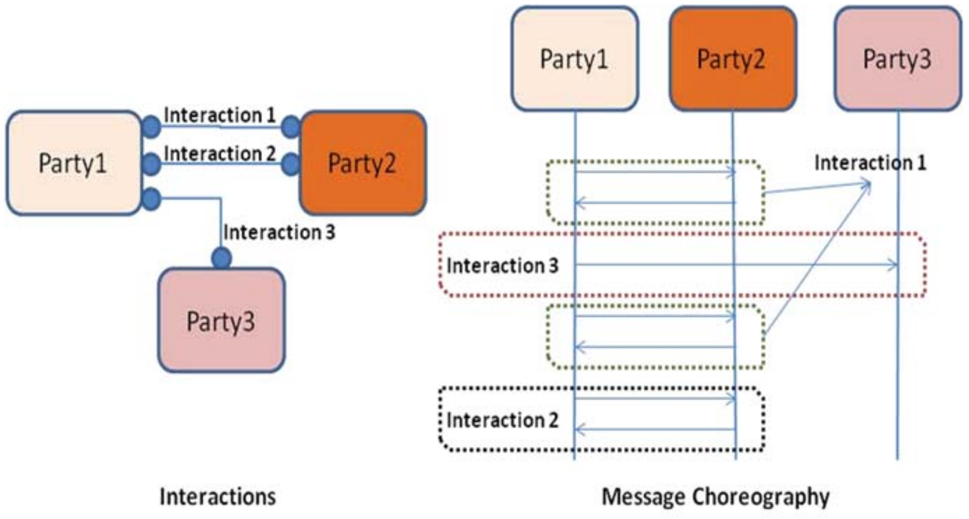


Figure 4.6: Interaction Concept

In the *ConfigurationGroup* element, the Test Designer should list all the interactions among parties by using a *CPA (CollaborationProfileAgreement)* element for each of them. There is an Interaction between two parties if there are one or more message exchanges among them in the test scenario. The Interaction concept does not have one-to-one correspondence with message exchanges; more than one message exchange can belong to same interaction. However, when there are two groups of message exchanges with different protocols, then two interactions should be defined between those parties. The Figure 4.6 illustrates a sample configuration and message choreography among systems.

A *CPA* element is defined for each interaction. The CPA constructs are used to prompt the SUTs so that they can provide their configuration values (e.g. communication related parameters such as the endpoints and the ports) used in the transaction. In the CPA element, the names of the parties involved in the interaction (i.e. party names are declared in ActorUnderTest and TestCaseCoordinator elements) are provided. In addition, a unique identifier is assigned for each interaction. The operational semantics of the CPA element is as follows:

- If both of the parties of the interaction are Simulated Actors (SAs), the configuration parameters of these parties are generated automatically by the TestBATN engine.
- If one of the parties of the interaction is a SA, the configuration parameters of that actor are generated automatically by the TestBATN engine. The Human Test Driver of the SUT playing the corresponding Actor Under Test (AUT) is prompted to obtain the SUT's configuration and show the SA's configuration.
- If both parties of the interaction are AUTs, since the TestBATN will play the proxy role, the configuration parameters for the proxy are automatically generated. So the Human Test Drivers of the SUTs playing the corresponding AUTs are prompted with TestBATN's proxy configurations. At the same time, the TestBATN framework needs the SUTs' configurations and therefore the SUTs are prompted to provide their configurations to the TestBATN Framework .

The configuration information are stored and indexed for each interaction.

4.2.3 Preliminary Scenario Establishment and Related Instructions

As mentioned earlier, in the Preliminary Scenario Establishment phase, the scenario requirements of the test scenario are presented to the Human Test Drivers. The Test Designer may intentionally give the responsibility to determine certain values to a specific Human Test Driver of a SUT. When a Human Test Driver specifies some of the initial test values, the other parties (Human Test Drivers of the SUTs) in the scenario are immediately notified.

These scenario requirements are defined through a list of *PreliminaryData* instructions in the *ConfigurationGroup* element. Each *PreliminaryData* element includes an information item with its scenario requirement description. The information item is bound to a variable either to retrieve the value and show it to the Human Test Drivers or to obtain a value and store it into the variable. This action type is defined with an attribute whose value can be “show” or “request”. The Test Designer should provide a party name so that the scenario requirement is only shown to or requested from the Human Test Driver of that party. A special value “ALL” for party name in “show” type actions will make the system to show the scenario requirement to all parties.

For the generation of random test data during the test case execution, the TestBATN framework defines the “Value Initiator Adaptor” Interface and through the *InitiateValue* instruction uses the value initiator adaptors. The *InitiateValue* instruction is used in Variable definitions and when processed during the execution, it assigns a random value to the variable. Generally, the purpose for this instruction is to generate different scenario requirements for every instance of the test case execution. The method that the adaptors use in generating random values is not important for the TestBATN framework. The Test Designer only deals with the input and configuration parameters of the adaptor according to their semantics described in the descriptor of the adaptor. A simple value initiator may get a list of values as input and randomly choose one of the values as return value.

4.2.4 Test Steps and Organizing the Test Flow

After all these preliminary steps, the test execution continues with the test steps that define the real actions in the test scenario. From the TestBATN perspective, a Test Step defines a single action which changes the state of the test instance context and ends with a result which can be “success”, “failure” or “undefined” together with a detailed test report. The Test Steps are further categorized into two classes in terms of the perception of the test scenario from the Test Drivers’ and Test Designers’ perspective. The first class includes the skeleton steps (the majority of test steps; messaging steps, validation steps, user interaction steps, and flow steps) which are presented to the Human Test Drivers as the test scenario steps. The second class only includes the auxiliary data processing steps (the Assign steps) which are hidden from the Human Test Drivers. Test Designer should provide a user-oriented narrative description for each step in the first class since they will be presented to the Human Test Drivers as scenario steps.

The TestBATN framework defines a presentation model for the test scenario to illustrate the sequencing and the test flow. After the preliminary phases, the TestBATN Engine generates the presentation model from the test case definition and sends it to the TestBATN Control and Monitoring GUI. The TestBATN Control and Monitoring GUI may present this model to the Human Test Drivers in different formats. Currently it supports a format which represents the steps in a sequence with some special numbering for the flow. A graphical presentation with sequence or actor diagrams is considered as a future work. The presentation model is also used as an interface between the TestBATN Engine and the TestBATN Control and Monitoring GUI in order to achieve real time monitoring of the execution.

For reusability and organizing the flow among the test steps, *ThreadGroup* and *Thread* constructs are used in the TestBATN framework. A *Thread*, similar to the thread concept of the programming languages, is a group of test step definitions which will be executed sequentially. Furthermore, since Threads are atomic entities in terms of functionality, the Test Designer may reuse them (like calling a function) in the test case definition in different part of the scenario. A *ThreadGroup*, on the other hand, consists of a list of threads which will be executed concurrently. Execution of a ThreadGroup is completed when the executions of all the threads defined under the ThreadGroup are completed.

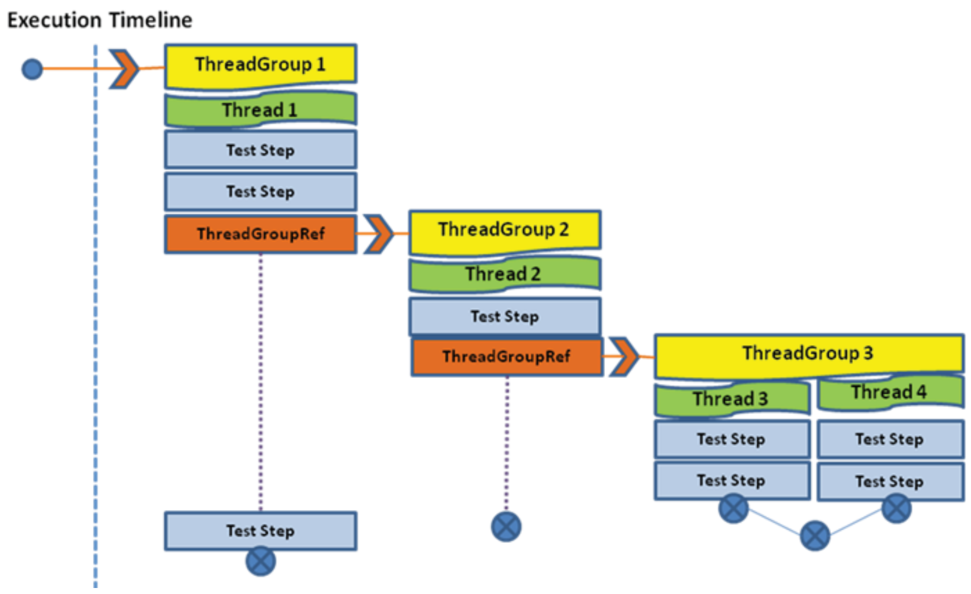


Figure 4.7: Execution Flow Among ThreadGroups

The execution flow among the ThreadGroups is handled by *ThreadGroupRef* instructions that can reside in Thread definitions. The Figure 4.7 illustrates the execution flow among thread groups for a sample test case definition. As shown in the figure, theThreadGroupRef instructions may be an intermediate step or the last step of a Thread definition. With these instructions, the execution continues with the referenced ThreadGroup and when it ends, the next step in the Thread definition is executed.

The *ThreadGroupRef* instruction is also used to design repetitive tasks in the scenario. When the instruction refers a *ThreadGroup* which is on the current execution flow, then the execution semantics is similar to GOTO instructions in the programming language. In other words, the execution flow is stopped at that point and execution of the referenced *ThreadGroup* is restarted. There are the following restrictions for the *ThreadGroupRef* utilization for repetitive steps:

- The *ThreadGroupRef* instruction, that will lead a repetition in the scenario, should be the last step of the parent construct.
- Inside the concurrent execution paths, it is forbidden to refer a ThreadGroup that leads the concurrency, or an upper *ThreadGroup* in the path.

For each repetition of a test step, a separate report and verdict is generated and presented to the Human Test Drivers.

4.2.5 Messaging Interfaces and Instructions

Within a conformance test scenario, sending messages to or receiving messages from the SUT is the most basic ability of a test framework. Similarly, for interoperability scenarios, the test framework should have the ability to listen the messaging between SUTs. In the TestBATN framework, there is a specific instruction to provide each of these abilities.

The *ReceiveMessage* element in a test case definition is used for a step when a SUT (playing a party role defined as Actor Under Test) is expected to send a message to a Simulated Actor (SA). Similarly the *SendMessage* instruction is used for the test steps where a SA is expected to send a message to a SUT. Finally, the *ListenMessage* instruction is used for the steps when a SUT is expected to send a message to another SUT and the Test Designer wants to listen to the messaging so that he can use it in further steps.

As mentioned earlier, the TesBATN framework is designed to support different messaging and transport protocols with the adoption of a modular approach; messaging interfaces and pluggable adaptors. In B2B scenarios, the transport protocols can be connection oriented like TCP/IP or DCCP or connectionless like UDP, IP or ICMP. In order to support connection oriented transport protocols the concept, called message transaction , is introduced in TestBATN TDL. A transaction is the group of message exchanges between two parties over the same connection. For example, in this way by stating that a ReceiveMessage and SendMessage are in the same transaction, the Test Designer can setup a part which can handle a connection oriented request-response interaction (e.g. HTTP request-response). Likewise, two ListenMessage instructions can be bound to the same transaction so that request-response interaction between two SUTs can be handled. The messaging instructions that belong to the same transaction do not have to follow each other in the test step sequence. The Test Designer can setup other test steps between them.

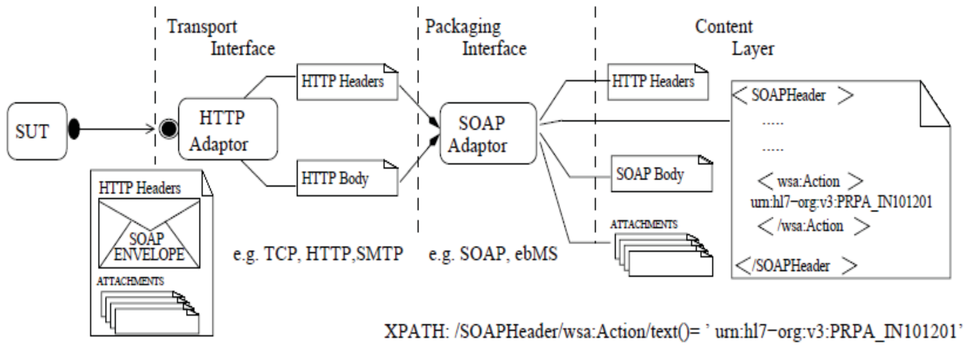


Figure 4.8: The TestBATN Approach for Messaging in Testing

The generic methodology for the interfaces, modular instructions and utilizing pluggable adaptors are described before. For messaging instructions, the semantics is the same as shown in Figure 4.4; the Test Designer states the name of the transport and packaging layer adaptors for the instruction and provides the input, output and configuration parameters. From functional point of view, the approach for ReceiveMessage instruction is illustrated in Figure 4.8. For this instruction, the Test Designer chooses an HTTPAdaptor for the Transport Interface and a SOAPAdaptor for the Packaging Interface to support a communication with SOAP over HTTP. From the adaptor descriptor files of these adaptors, the Test Designer knows that the HTTPAdaptor, after receiving the message, breaks the message into two fragments. It is also known that the SOAPAdaptor accepts these inputs and produces four message fragments where content of the first fragment is the HTTP Header, the second is the SOAP Header, the third is the SOAPBody, and last one being the attachments. The adaptor descriptors also describe the data types (either default data types, or user-defined object model types) for these output fragments. With this information, the Test Designer is able to assign each fragment to a suitable variable so that he can define test steps on the message content by using the variable reference.

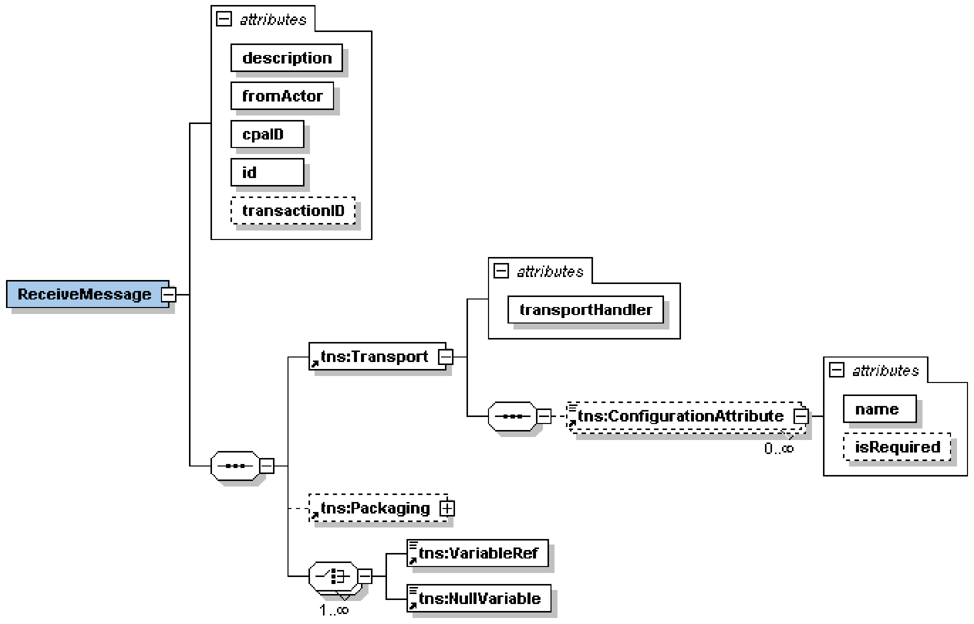


Figure 4.9: Data Model for ReceiveMessage Instruction

When it comes to the syntax for messaging instructions, the ReceiveMessage and ListenMessage are almost the same. The Figure 4.9 shows the data model for the ReceiveMessage instruction. Like the other test steps, there is an identifier (id) and description for the step. The fromActor attribute provides the name of the party that message is expected from. The cpaID attribute refers to the interaction that stores the configuration parameters for this message exchange. The transactionID attribute, as described above, provides the transaction identifier that this message step belongs to. The Transport element includes the instructions for Transport Interface; the identifier of the adaptor (the identifier that the adaptor is registered with to the framework), and the values for the configuration parameters. The content is similar for Packaging element which includes the definitions for the Packaging Interface. Finally, an ordered list of VariableRef elements defines the assignment of the output fragments to the referenced variables. The only difference for the ListenMessage instruction is the additional toActor attribute which states the name of the party that will receive the message.

Most of the data elements for the other messaging instructions are also used in the *SendMessage* instruction. However, for the *SendMessage* instruction, the Test Designer, instead of assigning adaptor outputs to the variables, needs to assign some content to the inputs of the adaptor for message generation. Instead of *VariableRef* elements, the *SetMessage* element is used for this purpose. The TestBATN framework defines two ways for Test Designers to provide the content for *SendMessage* instruction. The first one is providing a reference to a variable, similar to the assignment in the ReceiveMessage case, for an input defined for the adaptor. In this case, the content stored in the variable is passed as one of the inputs of the selected adaptor. The second way is providing a relative path of a file which includes the content. Such files are called Content Files in the TestBATN framework and deployed together with the test case definition in the test case deployment package.

For dynamic message content creation in test scenarios, the TestBATN framework provides a feature called *placeholders*. Placeholders are used in the *Content Files*, in which case the files are called *Message Template Files*. The placeholders simply act as variable references placed inside the *Message Template Files* so that the run-time values of the variables are replaced with that of the placeholder when the *Message Template Files* are loaded at the execution time. Thus, the placeholders make information gathered during the test execution available to message or document generation. The *Message Template Files* and the placeholders can only be used for textual content formats (e.g. XML, EDI, etc). For binary contents or other formats, the Test Designer should provide a *Content File* with the original content and the format. In this case, the modifications on the content can be done with further data processing test steps on the corresponding object model.

4.2.6 Expressions and Data Processing Instructions

The *Expression* construct and the XPATH based expression language for the content of the construct are used to facilitate the manipulation and processing of data stored in the variables. The construct is also used in other test step instructions like *Assign* or *TestAssertion*.

The output from the execution of an *Expression* definition is a data item where the data type for the information is one of the default data types defined in TestBATN framework, or a user-defined object model types. Since the Expression construct is used inside the other test step instructions, the expected data type for the output of the expression execution is known at the design time. Therefore, the Test Designer should be careful about the Expressions since the test case will be marked as invalid when the type of the output from expression execution does not match with the expected type, or a conversion between data types are not possible.

The *Expression* construct consists of the namespaceBindings attribute and the expression itself. The *namespaceBindings* attribute defines the bindings of the prefixes used in the expression to namespaces. Namespaces can be related with data (e.g. namespace of an XML element, namespaces for user-defined object data models) or functions used in the expressions.

```
expression        ::= XPATH-expr
                        | variable-exp  XPATH-expr
                        | variable-exp
                        | URL-exp
                        | NULL
variable-exp      ::= '$' NCName [array-ind-expr]
array-ind-expr    ::= '{' Number '}'
URL-exp           ::= '#' RelativeLocationPath
NULL              ::= "null"
```

Figure 4.10: BNF for TestBATN Expression Language

The TestBATN expression language is a simple extension on XPATH 1.0 expression language. The BNF for the expression language is shown in the Figure 4.10. As shown in the figure, the expressions can be in one of the following three formats in addition to the original XPath expression format.

- Variable-Expressions: These are expressions that start with a variable reference and continue with an XPath expression.

- URL-Expressions: These expressions provide the relative path for the *Content File* where the content is returned as output of the expression.

- NULL-Expression: This is simply the NULL value of the framework.

$patient	Returns the value of the variable ‘patient’.
$patients{2}	Returns the value of the second element in the list type variable ‘patients’.
$examination/hl7:diagnosis /text()	The value returned when the XPath expression ‘/hl7: diagnosis /text()’ is executed on the value of the ‘examination’ variable (‘hl7’ is the prefix for the namespace of the ‘diagnosis’ element).
js:isBefore($date1, $date2)	The value returned when the java script function ‘isBefore’ (java script functions can be defined in an external file and the path of the file is provided in javaScriptURI attribute in test case definition) is executed on the values of variable ‘date1’ and ‘date2’.
srdc_math:sin($angle)	The value returned when the function ‘sin’ is executed on the value of the variable ‘angle’. The ‘srdc_math’ is the prefix that is bound to a namespace in the *namespaceBindigns* definition. The namespace should be the namespace of a function library deployed to the framework.
#./SampleContent1.xml	The content of the file ‘SampleContent1.xml’

Figure 4.11: Sample TestBATN Expressions

The operational semantics for the expression language is illustrated in the Figure 4.11 through examples. In order to use functions in the expressions, the *Function Library* which implements the functions should be deployed to the framework. Like the other modular adaptors, *Function Library* deployment packages consist of the implementation for the functions and a Function Library Descriptor. The Function Library Descriptor describes the signature for each function; the name of the function, the data type of the return value, and the data types for input parameters. In addition, it may also include the narrative descriptions about the functions. The Function Library Descriptors are used to help Test Designers in test case design. The unique identifier for the Function Libraries in the framework is the namespace provided in the deployment phase. The functions in the library can be used in expressions by using this namespace in the namespaceBindings attribute of the Expression construct.

One of the instructions that the *Expression* construct is used for is the *Assign* instruction. The *Assign* instructions are viewed as internal test steps (hidden from Human Test Drivers) and are used to change the content of a variable. For the *Assign* instruction, the Test Designer devises an Expression construct where the returned value from the expression is stored or appended (for list type variables) to the variable.

4.2.7 Test Assertion Interfaces and Instructions

The *TestAssertion* instructions are used for semantic and syntactic validations of the information received from SUTs by messages or through user interactions. Each Test Assertion step corresponds to a set of constraints or requirements of the standard or interoperability profile that the SUTs are expected to conform. For each of these constraints, there can be a sub-instruction; *VerifyContent* or *ValidateContent* where different testing methodologies and adaptors can be used.

The *ValidateContent* is the test instruction which performs a syntactic validation against a structured schema or rules. It is one of the modular instructions and the adaptors implementing the "Validation Adaptor Interface" will be used for the instruction. For the instruction, the TestDesigner plugs-in a suitable adaptor (provides the identifier of the adaptor in validationAdaptor attribute), provides a schema for validation, and writes an Expression construct which will return the content to be validated.

The complementary instruction, the *VerifyContent*, aims at semantic validation and can be used for more complex testing adaptors and methodologies. The adaptors for this instruction should implement the Verification Adaptor Interface. The Verification Adaptors may organize their testing capabilities in different testing functions. These functions should be described in the Adaptor Descriptor. The Test Designer, by analyzing the function definitions, can select the suitable function (provides the name of the function with verificationFunction attribute of the instruction) and provides the input parameters of the function. The inputs are provided by an ordered list of Expression constructs where the return values for the expressions are the input for the adaptor.

Both the *ValidateContent* and the *VerifyContent* adaptors should produce a report and a verdict regarding the performed validation or verification. Report formats are specific to adaptors and can be textual or in XML format. While presenting the report to the Human Test Drivers, the TestBATN Control and Monitoring GUI uses the original format. However, the adaptor implementers, if their report format is XML, can provide a XSLT file which will be used to transform the XML into a presentation format.

The overall verdict for a *TestAssertion* step is calculated from the individual verdicts of each of the *ValidateContent* or the *VerifyContent* steps. This verdict is also used to make alternative execution flows at run time with *WhenTrue* and *WhenFalse* constructs. In this way, the alternative paths in the test scenario can be handled.

4.2.8 User Interaction Instructions

The TestBATN framework enables the interaction of the test execution with the Human Test Drivers in the preliminary phases and during the run-time monitoring. However, there will be some cases where an interaction is needed in the middle of the test scenario or where the test purpose can only be achieved by asking some questions to the Human Test Drivers about the executed test. The *RequestTestData* instruction is used for this purpose and can be used anywhere in the scenario.

With *RequestTestData* instruction, the Test Designer may prepare a list of questions for the Human Test Driver of a SUT where each question requests a specific value about the messages or scenario outcomes. The toActor attribute of the construct is used to indicate the party name that will answer the requests. Each question is bound to a variable reference so that the values obtained from the Human Test Driver are stored into the variables to facilitate further operations on them. While processing this instruction, the TestBATN engine will forward the questions to the TestBATN Control and Monitoring GUI of the selected party and the data requests are presented to the related Human Test Driver.

4.3 Test Design and Framework Management

The TestBATN framework defines the following phases for the design and definition of conformance and interoperability test cases:

- Test Plan: Any organization that wishes to build a test program for a standard or profile, first needs to define its objectives, the validation program and the conformance and the interoperability criteria. Standards or profiles may define different criteria or basis for conformance. As an example, some standards take the application roles that they define in their normative business processes as the basis for their conformance criteria.

Therefore, a vendor, when making a conformance claim, specifies the application roles that its product supports for a specified business process. Other standards, for example, may choose transaction based conformance claims. Furthermore, some standards may define conformance or interoperability levels where each has different criteria. All these details should be specified and documented as the first phase.

- Test Requirements Identification: In this phase, the specialists of the standard identify the test requirements for the target conformance or interoperability levels. These requirements should refer to the specification document that can be presented by a narrative text, optionally with some supplementary test material (e.g. schemas, test assertion with some expression format).

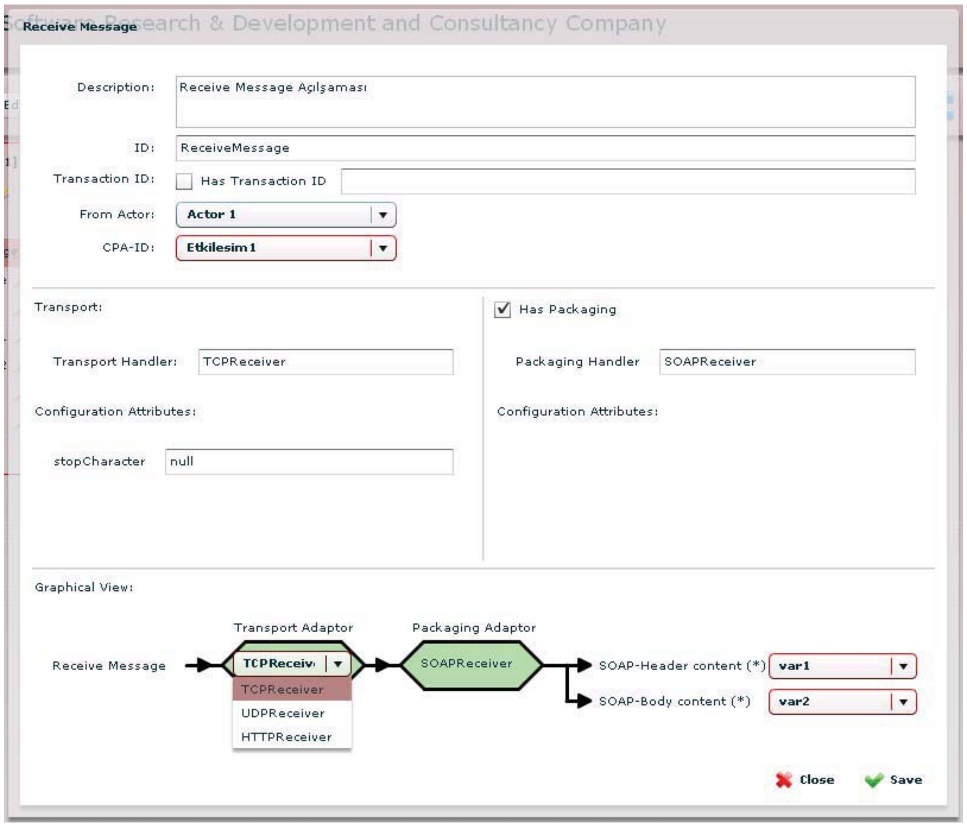
- Test Case Design: In this phase, test designers design test cases where each of them may cover a set of test requirements identified in the second phase. In this way, a set of test cases covering all the test requirements for a conformance or an interoperability level can be developed. Vendors claiming conformance or interoperability to that level then should run these test cases. The design process includes the design of a test harness by using the test instructions and pluggable modules provided by TestBATN.

- Development and Deployment of Pluggable Adaptors: As mentioned before, the TestBATN framework is based on a modular approach to support any eBusiness standard. Therefore, in the design phase, the test designer can select the suitable pluggable adaptors (from the modules that are deployed into the framework for the specified interface for the test instruction) and the module will support the execution of a specific test instruction. The desired adaptor may already exist in the framework perhaps developed within the scope of another test program of another standard and can be reused. Otherwise, it should be developed and deployed into the framework. In this phase, the test designer or the person in charge of these issues should develop the necessary modules. This phase also includes the testing, approval and deployment of the module to the framework.

- Test Case Validations: Before serving the test case definition to the users of the framework, the test case should be validated to see the errors in the definition or missing parts to achieve the test purpose.

- Test Case and Test Suite Deployment: When the test case and test suite definitions are ready, they should be deployed to the framework to serve vendors for their conformance and interoperability testing.

The TestBATN framework provides supplementary tools to help the test designers and implementers in these phases. One of them is the TestBATN TestDesigner GUI which supports the test designers graphically to design and implement test case and test suite definitions. It provides the ability to examine the pluggable adaptors, the data models, and the function libraries and facilitate to use them in the test case definition mostly in drag-and-drop manner. The TestBATN TestDesigner GUI is also designed to be modular and scalable. It performs all its operations by processing the Adaptor Descriptors, TestBATN TDL construct definitions, and the configuration files.


Receive Message

Description: Receive Message Açılşaması

ID: ReceiveMessage

Transaction ID: Has Transaction ID

From Actor: Actor 1

CPA-ID: Etkilesim1

Transport:

Transport Handler: TCPReceiver

Configuration Attributes:

stopCharacter: null

[✓] Has Packaging

Packaging Handler: SOAPReceiver

Configuration Attributes:

Graphical View:

Figure 4.12: A Snapshot from TestBATN TestDesigner GUI

The TestBATN TestDesigner GUI also provides a two phase test design approach where the first phase is the establishment of the skeleton of the test scenario; actors and message choreography. A Test Designer can graphically draw a business process which will then be transformed into TestBATN test case definition automatically. In the second phase, the Test Designer, again by using the TestDesign GUI's facilities can setup other test instructions. The Figure 4.12 shows a snapshot from the GUI for the ReceiveMessage instruction.

The TestBATN Framework Management GUI, on the other hand, helps the administrators and the other responsible user roles to maintain and manage the materials, user profiles, and system profiles deployed or registered into the framework. The main feature is the deployment of the pluggable, adaptors, function libraries and object data models.

CHAPTER 5

CASE STUDIES WITH TESTBATN

After the development of first prototype, TestBATN framework is used for conformance and interoperability testing of some standards and specifications. From these experiments, the most important is the development of test suites for the specifications of Turkish National Health Information System (NHIS). TestBATN with these test suites are used to test all medical information systems in Turkey to determine their conformance and interoperability with NHIS specifications and help them to find their non-conformant parts in their system. An experimental test suite is also developed in TestBATN for some IHE IT Infrastructure profiles to demonstrate the capability of TestBATN to cover testing requirements of IHE profiles. Finally, test suites are developed for IHE Radiology technical framework profiles which are based on Digital Imaging and Communication in Medicine (DICOM) standard which is different from other standards as the content is not XML and has a specific network protocol which provides different challenges and feedback for testing capabilities of TestBATN.

In this chapter, a summary of these experiments and how TestBATN capabilities are used to develop these test suites will be described.

5.1 Testing the Conformance and Interoperability of NHIS to Turkey's HL7 Profile

The current version of the NHIS, Turkey provides 23 Web Services, each of which is specialized to a specific Minimum Health Data Set Transmission as described in [54]. In order to guarantee the interoperability, the MoH, Turkey published an Implementation/Integration/Interoperability Profile [55]for vendors of Family Medicine Information System (FMISs) and Hospital Information Systems (HISs). This profile is based on the following national and international standards and specifications:

- For transport protocol, HL7 Web Services Profile [56] is used. For security, WS-Security [57] Username Token over SSL is required for conformance.
- The ”Transmission Schemas” are HL7 CDA R2 [58] compliant EHRs and each HL7 CDA section is a Minimum Health Data Set (MHDS) [59] which is formed from the data elements specified in National Health Data Dictionary (NHDD) [60].
- The Transmission Schemas are regarded as HL7 v3 messages and localized according to the national requirements. Localizations are represented as XML schemas for each Transmission Schema.
- Health Coding Reference Server (HCRS) [61] serves all coding systems in use in Turkey which is used in the data elements within the Minimum Health Data Sets. For some specific data elements, some other coding systems, like ICD-10 coding system [62], are specified for use.
- For each Transmission Schema and Minumum Health Data Set, several semantic Business Rules are defined to provide consistency among the values used in the data elements.
- In Turkey, every citizen has a unique identifier and these identifiers are maintained in a system called MERNIS (Central Demographics Management System) [63]. The patient id numbers in the messages are required to exist and be consistent with this system.
- In Turkey, every physician is registered to a system called Doctor Data Bank. The id numbers of the doctors in the messages should exist in this system.

The Integration Profile and its reference specifications present all the restrictions and the requirements for vendors to update or to develop their FMISs and HISs for a successful integration. However, without an extensive and effective testing process this may become difficult for vendors. Furthermore, only through testing, correct information exchange among these eHealth applications can be guaranteed and products can be certified.

By analyzing the requirements given in the integration profile and the eHealth market in Turkey in terms of the FMIS and HIS products, the following testing requirements are identified:

Basic Conformance Testing: The first step in the testing process is to test the ability of the HIS or FMIS systems to send valid "Transmission Schemas" in terms of syntactic and structural constraints. These tests include:

- Checking the ability of the systems to send HL7 Web Services Profile conformant SOAP (Web Service) messages,
- Checking the ability of the systems to send WS-Security Username-Token Profile conformant SOAP (Web Service) messages,
- Checking the ability of the systems to use the username and the password assigned to it correctly, as specified in the WS-Security Username-Token Profile in the corresponding SOAP header,
- Syntactic validations of "Transmission Schemas" sent by the systems against their corresponding XML schemas,
- Checking if the code systems and the codes used in the "Transmission Schemas" are valid, that is, checking whether the code system is one of the code systems specified in the integration profile and whether the code value is valid,
- Validation of the "Transmission Schemas" according to the corresponding business rules.

Functionality/Semantic Testing: Basic Conformance Testing guarantees the conformance of the "Transmission Schemas" sent by a system to the specified standards and specifications and hence it partially guarantees the MoH NHIS servers to accept the transmission and store its content to their database. However, it does not ensure that the information in the transmissions accurately represents the intentional semantics of the FMIS or HIS users (e.g. doctors, family practitioners). For example, a data element in the "Medical Examination MHDS" is called "Incident Type" which should contain values such as "Normal", "Emergency", or "Industrial Accident". For this data element, the following tests should be performed in order to guarantee the semantically valid transmissions:

- Testing whether the system provides all possible code values to its user for selection,
- Checking whether the system accurately packs the value selected by the user into the transmission,
- Testing whether the system (HIS or FMS) has the ability to render this value to its users. These testing requirements necessitate scenario based testing where vendors are given with a set of test scenarios and are requested to use their products' user interfaces to construct a transmission which is conformant with the given scenarios.

Interoperability Testing: Testing the conformance of applications systems to produce and consume the correct "Transmission Schemas" is necessary but not sufficient in deciding whether a FMIS or a HIS can properly be integrated into Turkey's NHIS. It is necessary to ensure that the selected options, bindings, and deployment settings of the implementations are compatible across partners.

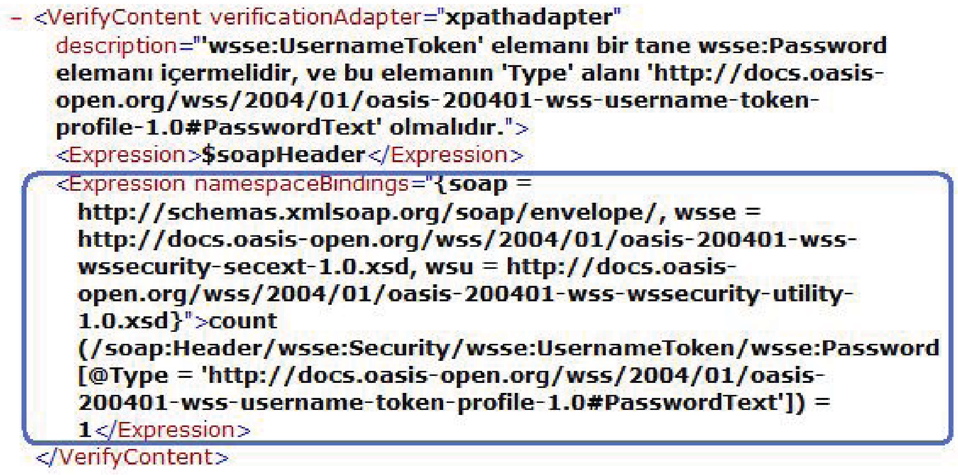
Appendix C provides a sample TestCase definition and other testing metarials used in the test case from the NHIS Test Suite. The next sections describe the different type of test cases in the NHIS test suite and how TestBATN abilities are used to perform conformance and interoperability tests based on NHIS specifications.

5.1.1 Test Scenarios for NHIS - Basic Conformance Testcases

For each "Transmission Schema", a basic conformance test case is written to test the conformance of FMISs and HISs to the requirements defined in the Integration Profile for the corresponding transmission. The following test steps describe these scenarios and the corresponding TestBATN framework functionalities used to execute them:

- SOAP Message Conformance: The first step is the messaging step where the SUTs (FMISs and HISs) are requested to send a transmission to the specified TestBATN Engine ports. By using the TestBATN messaging capability and choosing the "SOAP Message Adaptor" for this messaging step, the scenario is configured to accept only valid SOAP messages. Furthermore, by using the ability of TestBATN framework to define further configurations on Messaging Adaptors, the "SOAP Message Adaptor" is configured to test the further conformance criteria (HL7 Web Service Profile) defined for the transmission web services.

- WS-Security Username-Token Profile Conformance: The “SOAP Message Adaptor” partitions the message into three fragments; HTTP Headers, the SOAP Header, and the SOAP Body and TestBATN framework allows the test designer to use these fragments independently in the test steps that follow. WS-Security Username-Token Profile and further restrictions defined in the Integration Profile regulate the usage of “Username-Token” header in SOAP Header. In order to test these restrictions, semantic validation steps of the TestBATN framework are used. Each semantic validation step is actually the evaluation of an XPATH expression (corresponding to a single restriction) over TestBATN built-in “XPATH Validation Adaptor”. For example, an XPATH expression is written to test whether the message includes a single “Username” element in “UsernameToken” header element. Figure 5.1 illustrates a validation step with XPATH expressions to check the existence of wsse:Password element in wsse:UsernameToken security header.



```
- <VerifyContent verificationAdapter="xpathadapter"
    description="'wsse:UsernameToken' elemanı bir tane wsse:Password
    elemanı içermelidir, ve bu elemanın 'Type' alanı 'http://docs.oasis-
    open.org/wss/2004/01/oasis-200401-wss-username-token-
    profile-1.0#PasswordText' olmalıdır.">
    <Expression>$soapHeader</Expression>
    <Expression namespaceBindings="{soap =
      http://schemas.xmlsoap.org/soap/envelope/, wsse =
      http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
      wssecurity-secext-1.0.xsd, wsu = http://docs.oasis-
      open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
      1.0.xsd}">count
      (/soap:Header/wsse:Security/wsse:UsernameToken/wsse:Password
      [@Type = 'http://docs.oasis-open.org/wss/2004/01/oasis-
      200401-wss-username-token-profile-1.0#PasswordText']) =
      1</Expression>
  </VerifyContent>
```

Figure 5.1: Some Validation Steps for Username-Token Profile Conformance

- Username and Password Validation: The TestBATN user-interaction ability enables the test designer to obtain some preliminary information from the SUT user before test execution. By using this ability and its corresponding TestBATN construct “PreliminaryData” in the testcase definition, the username and the password that is assigned to the SUT are obtained before test begins. Then by exploiting two semantic validation steps with XPATH Validation, the username and password provided in preliminary steps are compared with the values given in the “UsernameToken” header of the message. This test is used for checking the ability of the SUT to use the correct login and password.

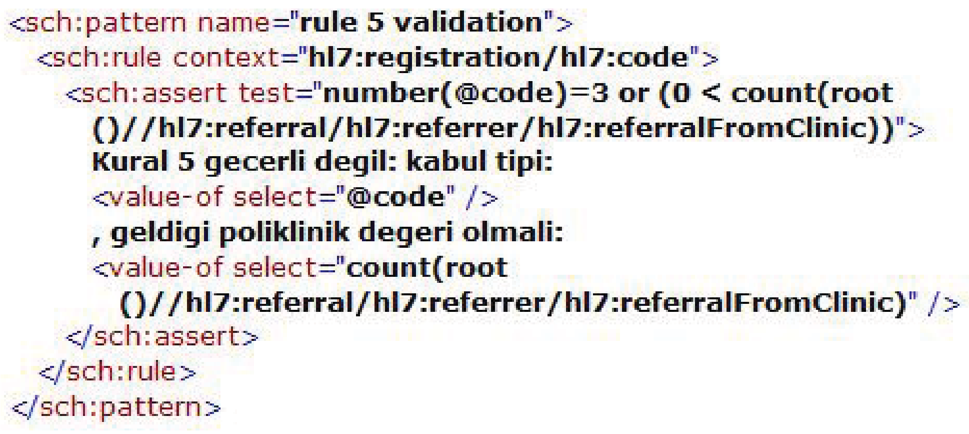
- Syntactic Validation of “Transmission Schemas”: This step is configured to use another built-in validation adaptor “XSD Validation Adaptor” and the input entries for the adaptor are the XML Schema of the corresponding transmission and the “Transmission Schema” received from the SUT (FMIS or HIS) in the SOAP Body.

- Validation of Coding Schemes and Codes: As mentioned previously, Health Coding Reference Server (HCRS) maintains all coding systems and codes in use in Turkey. This information is available through Web services from HCRS. In order to validate the codes and code systems used in a transmission, a new validation adaptor, SKRS Valisdator, is developed and plugged into the TestBATN framework. The SKRS Validator extracts all data elements with coded value type from the transmission and calls the HCRS services to validate these code values and the corresponding code systems.

- Validation Against Business Rules: Schematron definitions are used in specifying the business rules defined for the local constraints. For each transmission and for each MHDS in the transmission, the schematron rules corresponding to business rules for that transmission or MHDS are defined. In order to use these schematrons in the test execution, a validation step employing the built-in “Schematron Validation Adaptor” is used in the test case definition. Figure 4 illustrates a schematron rule, a part of the schematron for Patient Admission data set in Medical Examination transmission schema, which tests the business rule: “If the admission type is not “Other”; a value for the referrer clinic must be specified”.

5.1.2 Test Scenarios for NHIS - Interoperability Testcases

The TestBATN framework realizes the interoperability testing of Turkey’s NHIS for compliance with the Integration Profile defined by the MoH, Turkey, and the interoperability of the involved applications as follows:



```
<sch:pattern name="rule 5 validation">
  <sch:rule context="hl7:registration/hl7:code">
    <sch:assert test="number(@code)=3 or (0 < count(root
      ()//hl7:referral/hl7:referrer/hl7:referralFromClinic))">
      Kural 5 gecerli degil: kabul tipi:
      <value-of select="@code" />
      , geldigi poliklinik degeri olmali:
      <value-of select="count(root
        ()//hl7:referral/hl7:referrer/hl7:referralFromClinic)" />
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

Figure 5.2: A Schematron Rule to test a Business Rule

- First, it functions as a proxy between the SUTs (FMISs and HISs) and the MoH NHIS Server. The “Transmission Schemas” produced by the sender application are forwarded directly to the MoH Server by the TestBATN framework. Similary, the transmission responses produced by the MoH Server are intercepted and then forwarded to the SUTs.

- The framework internally stores all intercepted messages for further testing. Communication, Document and Business Process layer syntactic and semantic validations are performed.

- The TestBATN framework generates its own transmission response based on the profile constraints described in the MHDS. Its description language is equipped with the constructs (variable declarations, expressions and placeholders; flow constructs and corresponding messaging handlers) necessary for the Test Execution Engine to emulate the desired application behavior.

- As the final step, the transmission responses of the MoH Server are validated against the ones generated internally by the TestBATN framework. At this stage, any inconsistency is an indication of the MoH server’s noncompliance with the profiles and in such a case, applicable test reports are generated.

5.1.3 Test Scenarios for NHIS - Functionality Testcases

The TestBATN framework enables run-time customization of the scenario templates by means of its interaction capabilities with the SUT administrators (administrators of the FMISs, HISs and the MoH Servers) as shown in Figure 5. The TestBATN test description language supports user-interaction schemes to be defined either by its PreliminaryTestData or RequestTestData element constructs. In this manner, three different user-interaction schemes have been developed and used in the NHIS tests:

- Prior to scenario execution, the user is requested to fix the values for various test parameters: Some of the NHIS tests involve certain parameters to be fixed before the test execution by the SUT involved. The TestBATN framework uses the constructs called preliminary test variables for this purpose. For example, the healthcare institution and the author of the CDA document engaged in the "Transmission Schema" may need to be fixed in advance so that later on, the framework can use this information to perform various semantic tests such as i) the interface's ability to properly place this information in the related parts of the "Transmission Schema", ii) the FMIS's or HIS's compliance with the NHIS business rules. There is no limitation as to how preliminary test variables can be used within a test scenario, thanks to the capabilities of the TestBATN test description language.
- Prior to scenario execution, the scenario designer fixes some parameter values to test SUTs ability to work in a certain mode of operation and with the given control parameters: In contrast to the approach followed in (a), this time, the test designer imposes certain restrictions on how the SUT user should control its application behavior. The main objective behind this user-interaction scheme is to verify that the developed FMIS or the HIS is able to run in different modes of operation and that the application fulfills the requirements specific to that mode of operation. As an example, consider the case where the doctor fills out a detailed report regarding the referral of a patient to another healthcare institute. In this case, checking that the application processes the user input and that it places the correct ICD-10 code is one such test alternative.
- During scenario execution, the user is requested to provide feedback on the observed application behavior: It may be desirable to obtain feedback from the user during scenario execution for the purpose of assessing the SUT's rendering capabilities. In one of

the test scenarios for instance, the administrator of the FMIS/HIS is asked to provide the exact date/time of patient admittance. This information can be used in a validation step to ensure that the rendered date/time on the GUI of the FMIS/HIS is accurately used within the transmission.

5.1.4 Test Scenarios for NHIS - Complex Testcases

Some business rules defined for the NHIS impose semantic dependencies between different "Transmission Schema" instances. To be more specific, a business rule of the form: "No two different Vaccination Transmission Schema instances for the same patient may refer to the same Vaccination Code together with the same Vaccination Date/Time", is valid and needs to be tested accordingly. Such tests are adapted from the aforementioned approaches with some minor modifications on the previous scenarios:

- The first change involves extending the scenario with multiple ReceiveMessage constructs so as to receive as many Vaccination Transmission Schema instances as required per test scenario.
- The TestBATN test description language enables list-type variables to be declared. Capacities of list-type variables are dynamic, that is, they expand as new items are inserted. The scenario is modified such that the received documents are stored consecutively in the list.
- By utilizing the flow constructs of the language, the content of each received message are compared with all other messages stored in the list to see if the business rule is satisfied.

5.2 Interoperability Testing for some IHE IT Infrastructures Profiles

Conformance does not guarantee interoperability and in interoperability tests, software tools perform scenario based tests involving more than one application role and the interoperability test scenarios may involve complex cases where interactions are related. In order to provide insight to the requirements of interoperability, a test scnario is developed for a use case including three IHE Integration profiles combined to realize the sharing of EHRs, namely, the "IHE

Cross Enterprise Document Sharing (XDS)" [64] Profile, "IHE Cross-Enterprise Sharing of Medical Summaries (XDS-MS)" [65] Profile and the "Patient Identifier Cross-Referencing (PIX)"[66] Profile.

The basic idea of IHE XDS is to store healthcare documents in a registry/repository architecture to facilitate their sharing. IHE XDS is not concerned with document content; it only specifies metadata to facilitate the discovery of documents. Documents in IHE XDS may include any type of information in any standard format such as simple text, formatted text (e.g., HL7 CDA Release One), images (e.g., DICOM [67]) or structured and vocabulary coded clinical information (e.g., CDA Release Two, CEN ENV 13606 [68] or DICOM SR). Given this, to ensure the interoperability between the document sources and the document consumers, IHE publishes content profiles which specify the document format, the structure and the content. The IHE Cross-Enterprise Sharing of Medical Summaries (XDS-MS) Profile is one of the content profiles to automate the sharing of medical summaries between care providers. XDS-MS Profile uses the Actors and Transactions of IHE XDS; only the Document types used in XDS-MS are more specific medical summaries.

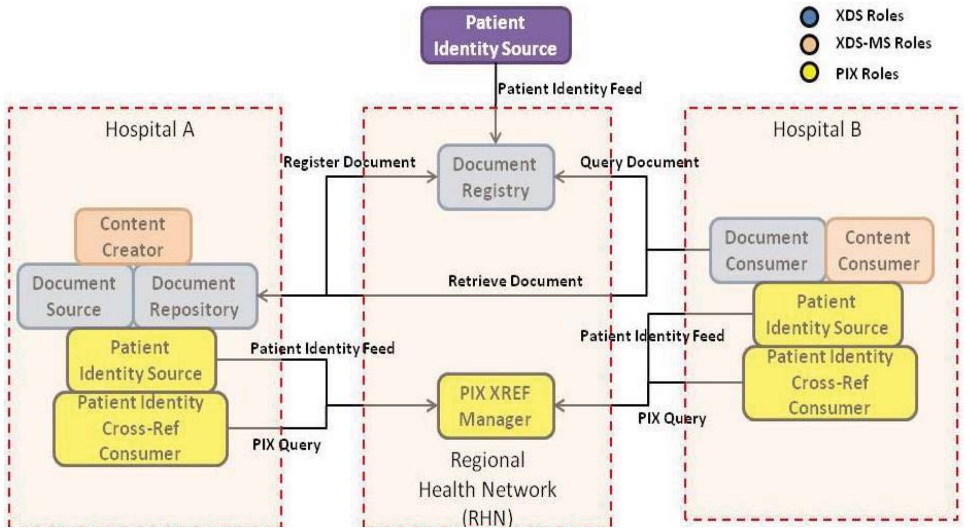


Figure 5.3: An Example Scenario with IHE profiles

IHE PIX, on the other hand, provides cross-referencing of patient identifiers from multiple Patient Identifier Domains. These patient identifiers can then be used by identity consumer systems to correlate information about a single patient from sources that use different patient identifiers. As shown in Figure 5.3, in this scenario, the Hospital A plays the “Consent Creator”, “Document Source”, “Document Repository”, “Patient Identity Source” and “Patient Identity Consumer” roles and the Hospital B plays the “Content Consumer”, “Document Consumer”, “Patient Identity Source” and “Patient Identity Consumer” roles. The objective of this scenario is to facilitate the sharing of medical summaries among hospitals via the “Regional Health Network (RHN)” which plays the “Document Registry” and “PIX XREF Manager” roles.

An interoperability test scenario for this use case will test three systems: two hospital information systems and one central regional health system, all of which claim conformance to the XDS, XDS-MS and PIX integration profiles. As testing requirements, first of all there are syntactic and semantic restrictions on the formats and payloads of the transactions shown in Figure 5.3. In addition, the systems should conform to message choreography and should realize the responsibilities of each role as a result of the transactions and the choreography.

In order to test this use case scenario, first it is necessary to exchange the configuration information of the systems so that they can send/receive message to/from each other. For example, the “Hospital A” system should know the network address of RHN in order to initiate the necessary interactions. After configuration, there is a need to inform all the parties about the initial test data. In the given scenario, the initial test data is the patient demographics information. However, the Test Designer may wish to design a more generic test scenario instead of restricting the tests to some specific patient information. For example, the Test Designer may appoint the administrator of the hospital information system, “Hospital A”, to be in charge of determining the patient information. So there is a requirement on how to inform all the related parties about the initial test data.

In this scenario, we assume that the systems have different patient identity domains. Therefore, each system should create a patient and assign an identifier to the patient in their system according to the demographic information given in the initial test data. When such patient identifiers are created, “Hospital A” and “Hospital B” should feed these identifiers to the RHN system (PIX XREF Manager role) according to the business process defined in the PIX profile. Similarly, as shown in Figure 1, another system which plays the Patient Identity Source role should feed the RHN system with the master patient identifier that will be used in RHN for this patient. In this scenario, we assume that the test framework will itself emulate the Patient Identity Source role. This step requires the template “Patient Identity Feed” message conforming to the HL7 v2 provided by the Test Designer to be tailored at run-time according to the semantics of the scenario. There should be mechanisms to use the values such as the patient demographics information initialized by the administrator of the “Hospital A” system at run-time and to generate random patient identifiers to feed the registry.

In the next phase, the scenario requires that the applications playing the Content Creator role should be able to produce valid medical summaries according to the XDS-MS profile. In order to test if the “Hospital A” system has this capability, the administrator of the system is requested to produce a medical summary document based on the scenario descriptions provided by the test writer. An example scenario description can be: “Patient is discharged from Hospital A with a discharge summary document”. Patient has allergy to penicillin and this information should be provided in the document in the Allergies section”. In order to test whether EHR system is capable of generating such a discharge summary document conformant with the XDS-MS profile, it is necessary to intercept this message and check if the document produced complies first with the profile (syntactically - e.g. through an XSD Adaptor) and then with the rule definition given above (semantically - e.g. through Schematron validation or XPath evaluation).

In the next step, the administrator of “Hospital B” queries RHN for all “Discharge Summary” documents of the given patient. The “Query Document” and its response messages need to be intercepted for further syntactic and semantic processing. More specifically, the query parameters should be tested to check whether they are set correctly for the document type. This is necessary to validate the conformance of the exchanged documents as well as to check whether the systems involved provide the expected functionalities.

Following the previous interaction, the scenario requires the administrator of “Hospital B” to select the document reference returned by the query and initiate a “Retrieve Document” transaction to retrieve the desired document. In this test step, it is necessary to have mechanisms to verify that the document is actually the one that has been registered. According to the XDS-MS profile, the “Content Consumer” role should properly render the CDA document received. The Test Designer should be able to request information from the system under test about the document content to check whether it was able to render the retrieved document.

5.2.1 Test Scenario for the IHE Use Case

Major test steps developed for this use case are summarized below:

- System configuration is the first step in a testing process. Each system should know the parameters of the others in order to configure its own system. TestBATN test description language has instructions that allow the Test Designers to specify the interactions among the systems explicitly. Through these instructions, the configuration information of the systems is exchanged over the user interfaces.
- There is a need to inform all the parties about the initial test data. In the given scenario, the initial test data are the patient identifier and the patient demographics information. The TestBATN test description language has a special construct PreliminaryTestData which instructs the TestBATN software to present such data to the related parties over the user interfaces. Once this information is made available, the related parties register the patient to their systems.
- The next phase of the scenario involves the "Patient Identity Feed" transactions from "Hospital A" and "Hospital B". The TestBATN "ListenMessage" construct intercepts the messages between two systems under test. When these messages are intercepted, validation instructions of the TestBATN framework are used to validate the message syntactically and semantically. For semantic validation, the content of the message, that is, the patient demographics information is compared with the information specified in the initial test data.
- In the next step, the test engine simulates the "Patient Identity Source" role and initiates the "Patient Identity Feed" transaction to feed the Document Registry with the identifier. The template "Patient Identity Feed" message provided by the test designer is tailored at run-time according to the semantics of the scenario. Placeholders given in the template document make this dynamic functionality possible. The test engine substitutes these placeholders with their run-time values such as the patient identifier and demographic information.
- After these steps, all identifiers provided by the systems should be correctly mapped in the RHN system as the XREF Manager role of the PIX profile requires. This ability of the RHN system can be tested in the next step when the "Hospital A" sends a PIX query

to XREF Manager to get the identifier used by the Document Registry for the patient. The result of the query is intercepted and the identifier returned by the XREF Manager is compared with the identifier that is generated and fed to registry in step 3.

- As mentioned earlier, the scenario requires that the applications playing the "Content Creator" role should be able to produce medical summary documents conformant with XDS-MS Profile according to the given scenario. In order to test this capability, the "Register Document" transaction is intercepted and the entries in the Allergies section are checked if there is an allergy entry about penicillin allergy as stated in the scenario description.

- In the next step, the administrator of "Hospital B" is requested to query RHN for all "discharge summary" documents of the given patient. In this step, the query parameters in the "Query Document" transaction are tested to check whether they are set correctly for the document type that is "discharge summary". In this way, not only the exchanged documents are validated for conformance, but also it is possible to determine if the parties provide the expected functionalities.

- Following the previous interaction, the administrator of "Hospital B" initiates a "Retrieve Document" transaction to retrieve the desired document. In this step, the Test Engine verifies that the document is actually the one that has been registered by using the special built in functions like hash comparison.

- According to the XDS-MS profile, the "Content Consumer" application role should properly render the CDA document received. Since the test designer is the producer of the document, by requesting information from the system under test about the document content through the RequestTestData construct and comparing the answers with the data contained in the variables through test assertions, the ability of the system under test to correctly render the document is tested.

5.3 Conformance Test Suite for IHE Scheduled Workflow Profile

The Scheduled Workflow Integration Profile (SWF) [69] defines the basic workflow within a radiology department in a hospital in order to establish the continuity and integrity of basic departmental imaging data. It is the backbone of IHE Radiology Technical Framework where other profiles define further transactions and restrictions on top of it. IHE SWF specifies a number of transactions that maintain the consistency of patient and digital image ordering information as well as providing the scheduling and imaging acquisition procedure steps.

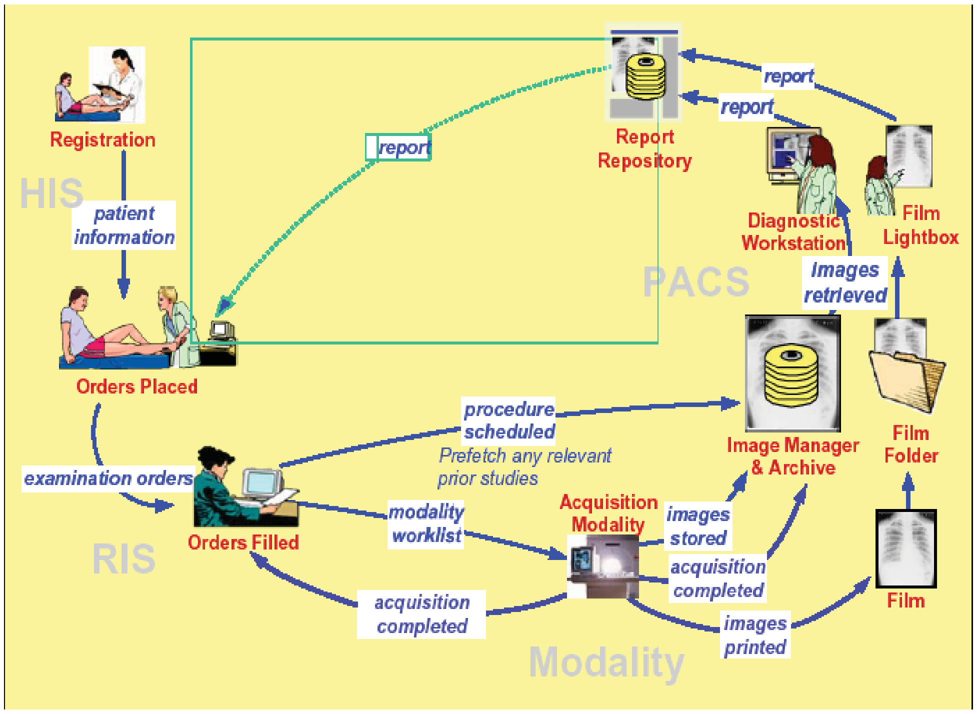


Figure 5.4: IHE SWF- Basic Workflow In a Radiology Department

The Figure 5.4 shows the basic workflow among hospital information system (HIS) and a radiology department; Radiology Information System (RIS), modalities (digital imaging devices; MR, CT, etc) and Picture Archiving Systems (PACS). The SWF profile provides the specification for this workflow where systems are defined as IHE actors and communications among these actors are specified as IHE transactions. The workflow starts with the patient registration to the hospital's registration module (IHE ADT actor) where registration data is sent (HL7 v2 ADT message) to both the department X who will make the imaging order (IHE Order Placer) and the radiology department (IHE DSS/Order Filler). Then the physician in the department X (IHE Order Placer) sends an imaging order (HL7 v2 ORM Message) to the radiology department (IHE DSS/Order Filler) after examining the patient (ex: Chest X-ray). The nurse operating on the RIS system (IHE DSS/Order Filler) sees the order and schedules it for the suitable imaging device for a certain time. A notification (HL7 v2 ORM Message) is sent to PACS system (IHE Image Manager-Archive) for the scheduling operation. When the patient goes to the medical imaging room, the technician queries (DICOM Modality Worklist Query) the RIS system (IHE DSS/Order Filler) to get the details of the operation by using the imaging device (IHE Acquisition Modality). After the acquisition of images is started and completed, the modality sends notifications (DICOM Modality Performed Procedure Step messages) to RIS (IHE DSS/OF) and PACS (IHE Image Manager-Archive) systems. The images are also sent (DICOM Image Storage message) to PACS system to store them in the archive. The radiologist responsible with the report, by using his workstation (IHE Image Display and IHE Evidence Creator), queries and retrieves (DICOM Query/Retrieve) images to examine them and write a report for the patient. The scenario finalizes with this operation. The access of referring doctor to images and radiology reports are specified with other IHE Radiology profiles. As seen from the scenario, the transactions are based on HL7v2 and DICOM standards where IHE defines further restrictions to ensure interoperability. In order to better describe the testing process and capabilities, a brief summary of DICOM standard will be given in the next section.

5.3.1 Brief Introduction to DICOM

DICOM [67] is known as the de facto standard for medical image communication. The standard defines data structures and services for the vendor-independent exchange of medical images and related information. It is being developed by medical industry and medical professional organizations under the umbrella of the National Electrical Manufacturers Association (NEMA) [70].

Today, virtually all imaging devices that are used in radiology, such as CT, MRI, Ultrasound, RF, and other digital rooms, support the DICOM standard for the exchange of images and related information. Imaging devices for other domains, such as endoscopy, pathology, ophthalmology and dermatology have also started implementing the DICOM standard, because there is an increasing need to integrate all these images into the patient master record (e.g. EHR).

The DICOM Standard facilitates interoperability of medical imaging equipment by specifying:

- A set of protocols for network communications to be followed by the devices claiming conformance to the Standard.

- The syntax and semantics of the commands and associated information which can be exchanged using these protocols.

- A set of media storage services for media communication to be followed by devices claiming conformance to the standard, as well as a File Format and a medical directory structure to facilitate access to the images and related information stored on the interchange media.

- The information that must be supplied with an implementation for which conformance to the Standard is claimed.

In DICOM general communication model, "Service Class Specifications" are defined to provide the operations among different systems. The message content for these objects is defined with Information Object Definitions. The standard also defines how the messages and images will be encoded and provides the data structures and data types for this encoding. Under this layer, images and other reports can be shared by other systems either online or offline by some media according to DICOM communication protocol.

Tag	Name	VR	VM
(0008,0020)	Study Date	DA	1

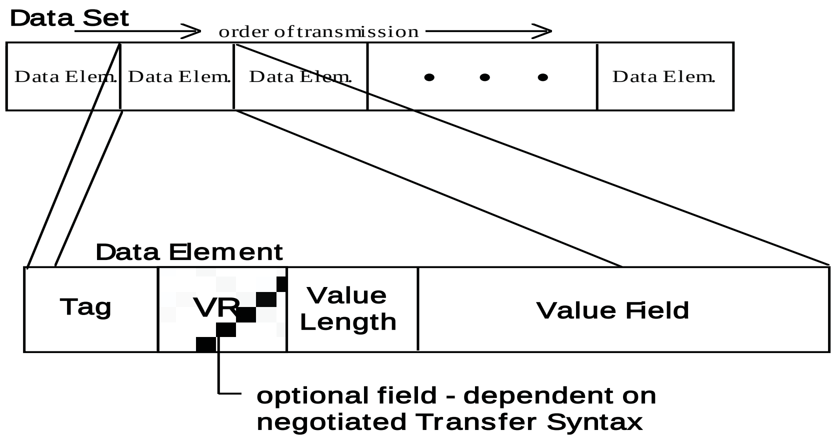


Figure 5.5: DICOM Data Set and Data Element

5.3.1.1 DICOM Information Model and Encoding

DICOM has an information model which differentiates it from other standards. The DICOM information model includes relationships between the DICOM objects as well as a common terminology. Information Object Definitions (IOD), which gives the definition of the information to be exchanged, is the main part of its information model. One can think of IODs as templates that are re-used over and over again when a modality generates a new image or other DICOM object. Each image type, and therefore information object, has specific characteristics. For example, a CT image requires different descriptors in the image header than an ultrasound image or an ophthalmology image. These templates are identified by unique identifiers, which are registered by the NEMA. The Attributes of an IOD describe the properties of a Real-World Object Instance. Related Attributes are grouped into Modules which represent a higher level of semantics.

A DICOM Message Service Element (DIMSE) Service Group specifies one or more operations/notifications which are applicable to an IOD. The union of an IOD and a DIMSE Service Group is called a Service-Object Pair (SOP) Class. The SOP Class definition contains the rules and semantics which may restrict the use of the services in the DIMSE Service Group and/or the Attributes of the IOD. For example, “CT Storage SOP Class” defines the message and data structures to transport a CT images from one system to another for storage. The top element, Service Class Specification contains a group of one or more SOP Classes related to a specific function which is to be accomplished by communicating Application Entities. A Service Class Specification also defines rules which allow implementations to state some pre-defined level of conformance to one or more SOP Classes.

In addition to abstract information model, DICOM also defines how this model will be represented and encoded syntactically. It defines some value types for the representation of values. For example, the type “DA (Date)” defines how date information will be encoded and represented. Similarly, “PN (Person Name)” defines how the name of a person will be represented.

DICOM uses a binary encoding with hierarchical lists of data elements identified by numerical tags and a complex DICOM-specific application-level network protocol. These hierarchical lists are called a Data Set. In other words, DICOM groups information into data sets. That means that a file of a chest X-Ray image, for example, actually contains the patient ID within the file, so that the image can never be separated from this information by mistake. A Data Set represents an instance of a real world Information Object. A Data Set is constructed of Data Elements. Data Elements contain the encoded values of object attributes. A Data Element is uniquely identified by a Data Element Tag. The Data Elements in a Data Set shall be ordered by increasing Data Element Tag Number and shall occur at most once in a Data Set. A Data Element is made up of fields; Data Element Tag, Value Length, Value Field, and Value Representation as shown in Figure 5.5. DICOM also defines a registry of data elements, DICOM data dictionary, where each data element in DICOM standard is defined and tag number is assigned. The Figure 5.5 shown how the “Study Date” element is defined in the DICOM data dictionary. Tag is the unique identifier for the element, VR specifies the data type and VM specifies the multiplicity of the element.

5.3.1.2 DICOM Message Exchange, Network Communication Protocol and DICOM Services

DICOM Network Communication Protocol is a proper subset of the services offered by the Open Systems Interconnection (OSI) Presentation Service (ISO 8822) [72] and of the OSI Association Control Service Element (ACSE) (ISO 8649) [71]. Therefore there is a negotiation phase before exchanging actual messages among systems. In the negotiation phase which is called the Association Negotiation, DICOM systems negotiate some DICOM specific parameters for the messaging; the transfer syntax (encoding of exchanged DICOM objects), abstract syntax (the class of the exchanged DICOM object, i.e. CT Image Storage).

After the association is established, the real message exchange starts. DICOM defines the basic message exchanges for the definition of real processes. These services are called DIMSE (DICOM Message Service Element) services. For example, the C-STORE service is used to request storage of a DICOM object and C-FIND is used to query DICOM objects. From the union of these basic service elements and IODs, the SOP class definitions are formed. Then from these SOP class definitions, the Service Class Specifications which defines the processes and choreographies for medical imaging in real life are formed. For example, the Query/Retrieve Service Class defines a service which supports the query of medical images where the query result is a list of image references which has specific properties as specified in the query. The C-FIND, C-MOVE, and C-GET are utilized in this service.

```
SOPCLASS "1.2.840.10008.5.1.4.1.1.2" "CT Image Storage SOP Class"

MODULE "Patient Module" M
(0x00100010,2,PN,1)                              "Patient's Name"
(0x00100020,2,LO,1)                              "Patient ID"
(0x00100030,2,DA,1)                              "Patient's Birth Date"
(0x00100040,2,CS,1,E,"F"
                  | "M"
                  | "O")                         "Patient's Sex"
(0x00100021,3,LO,1)                              "Issuer of Patient ID"
(0x00100032,3,TM,1)                              "Patient's Birth Time"
(0x00101000,3,LO,1:n)                            "Other Patient IDs"
(0x00101001,3,PN,1:n)                            "Other Patient Names"
(0x00102160,3,SH,1)                              "Ethnic Group"
(0x00104000,3,LT,1)                              "Patient Comments"
(0x00081120,3R,SQ,1,
 >(0x00081150,1C,UI,1,E,"1.2.840.10008.3.1.2.1.1")  "Referenced SOP Class UID"       : PRESENT ../0x00081120
 >(0x00081155,1C,UI,1)                           "Referenced SOP Instance UID"    : PRESENT ../0x00081120
)                                                "Referenced Patient Sequence"
```

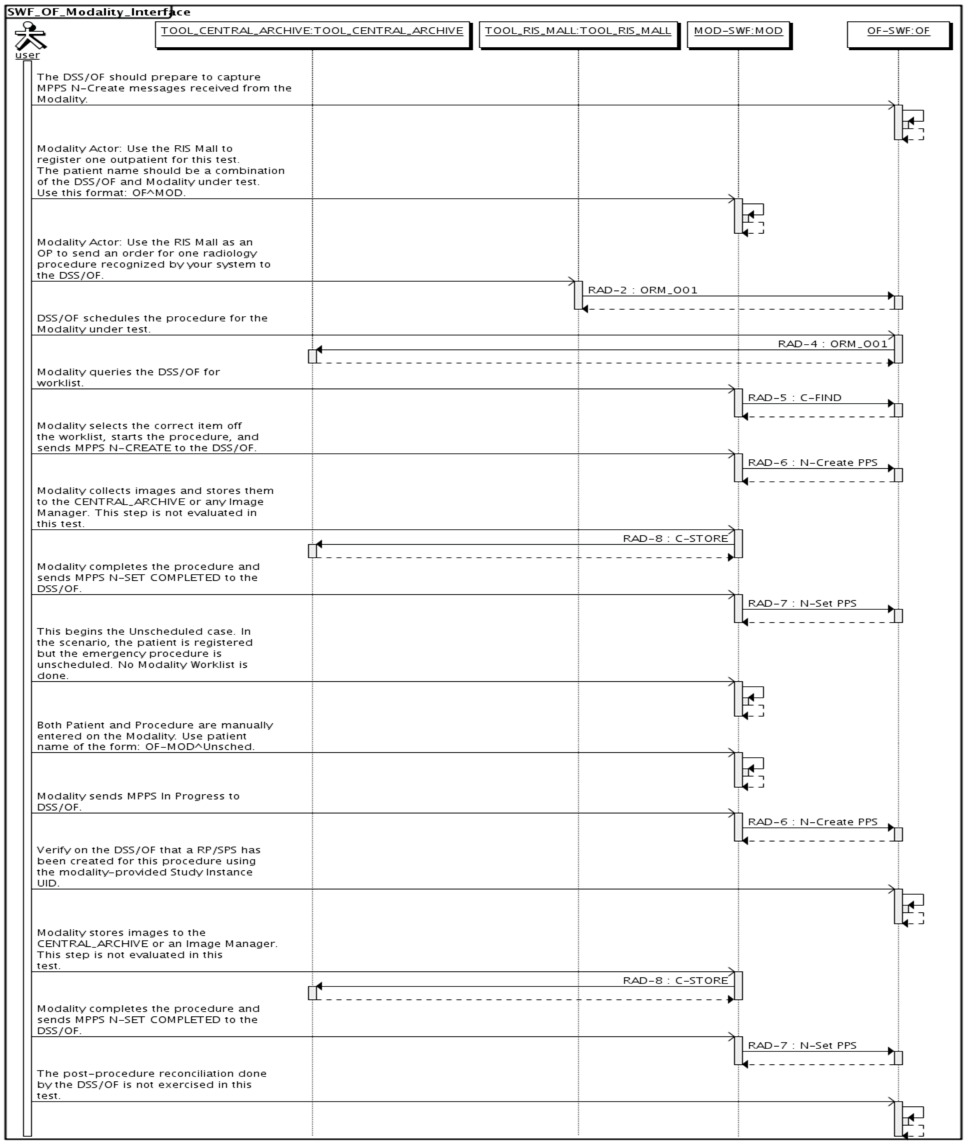
Figure 5.6: A Sample DICOM Validion Toolkit Schema

5.3.2 Conformance Test Suite for IHE SWF

IHE SWF is one of the most complex specifications of IHE that has 9 actors and 20 different transactions. In this case study, a complete conformance test suite is developed for the “Image Manager and Archive” (part of PACS system) and “DSS/Order Filler” (RIS systems) actors of IHE SWF. Test cases are grouped as follows as they are grouped in IHE’s annual test meetings, Connect-a-thons:

- *Conformance Test Cases for Each Transaction:* In order to perform focused and detailed testing on the conformance of actors to the specifications of each transaction, a number of test cases are developed for each transaction. For example, the DICOM based transaction RAD-5 Query Modality Worklist is used by modality to query the work list of operations that will be assigned to itself on the RIS system (IHE DSS/Order Filler). In order to test the DSS/Order Filler actor for its conformance to the transaction and ability to correctly responds to different type of queries with several query parameters, several test cases are developed. Each test case sends different type of queries with variety of query parameters and check if the query responses are conformant to RAD-5 transaction and responses are correct semantically. For the transaction testing 81 test cases are developed to test the all transactions that the DSS/Order Filler and Image Manager/Archive actors take place.

- *Workflow Test Cases:* The main objective of these tests is to check the conformance of the system to the requirements and restrictions specified for the actor role that the system is claiming to conform within the general workflow and sub workflows of the



The DSS/OF should prepare to capture MPPS N-Create messages received from the Modality.

Modality Actor: Use the RIS Mall to register one outpatient for this test. The patient name should be a combination of the DSS/OF and Modality under test. Use this format: OF^MOD.

Modality Actor: Use the RIS Mall as an OP to send an order for one radiology procedure recognized by your system to the DSS/OF.

DSS/OF schedules the procedure for the Modality under test.

Modality queries the DSS/OF for worklist.

Modality selects the correct item off the worklist, starts the procedure, and sends MPPS N-CREATE to the DSS/OF.

Modality collects images and stores them to the CENTRAL_ARCHIVE or any Image Manager. This step is not evaluated in this test.

Modality completes the procedure and sends MPPS N-SET COMPLETED to the DSS/OF.

This begins the Unscheduled case. In the scenario, the patient is registered but the emergency procedure is unscheduled. No Modality Worklist is done.

Both Patient and Procedure are manually entered on the Modality. Use patient name of the form: OF-MOD^Unsched.

Modality sends MPPS In Progress to DSS/OF.

Verify on the DSS/OF that a RP/SPS has been created for this procedure using the modality-provided Study Instance UID.

Modality stores images to the CENTRAL_ARCHIVE or an Image Manager. This step is not evaluated in this test.

Modality completes the procedure and sends MPPS N-SET COMPLETED to the DSS/OF.

The post-procedure reconciliation done by the DSS/OF is not exercised in this test.

Figure 5.7: IHE 2011 Connectathon SWF-OF-Modality Interface Test Scenario

profile. They test the responsibilities of the system under test regarding the relations between the different transactions defined in the workflow. For example, a test case is developed to test the conformance of the Image Manager/Archive playing system within the sub workflow among the DSS/Order Filler (RIS system), Acquisition Modality (Imaging Device) and the system under test; the Image Manager/Archive. The workflow test cases also include the basic conformance testing parts from the transaction conformance test cases in order to perform basic conformance testing on transactions that take place in the workflow. For the workflow tests, the 20 test scenarios which are used in IHE Connect-a-thons are developed as a TestBATN testcase.

The test cases are initially used by a HIS vendor in Turkey to prepare their RIS and PACS system for the 2011 IHE connect-a-thon in PISA. After a one-month period where the company runs the test cases and corrects the errors and non-conformant parts of their system that the test results show, their products pass the connect-a-thon tests and get the IHE certification for their system.

5.3.3 Preparing TestBATN Framework for DICOM Conformance Testing

5.3.3.1 Enhancements on TestBATN Network Layer

As mentioned in the summary of DICOM standard, DICOM is quite different and has much more complexity than other contemporary standards in eBusiness, eHealth and eGovernment. The reason is that different type of medical imaging devices with different technologies, i.e. Computer Tomography (CT), Magnetic Resonance Imaging (MR), Ultrasound Image (US), etc exist and the standard has to deal with all these image formats. From the conformance and interoperability testing perspective and the testing capabilities of TestBATN v1.0, the most challenging difference of DICOM standard was its communication protocol. The first version of TestBATN framework is designed to handle any type of communication protocol based on request/response message pattern on a single network connection which can handle many of the contemporary communication protocols like SOAP Web Services, Representational State Transfer (REST) services, simple HTTP Get/Post communications, and other simple TCP/IP and UDP protocols used in eBusiness, eHealth and many other domains. The assumption on the first version is that an established network connection is used only by single request-response exchange which is called TestBATN transaction. On the other hand, DICOM communication protocol requires association establishment on a tcp-connection and exchange of several messages in both direction on the same tcp-connection until one side releases the association and the connection by some special messages (ex: A-ABORT and A-RELEASE). As the DICOM communication protocol is a subset of OSI network communication model and is a very good example of connection oriented protocol, it is a very important step for TestBATN to reach the goal of being a generic framework; that is the ability to support any type of communication protocol. Therefore, the network layer of TestBATN framework is redesigned and enhanced with further capabilities with this DICOM based case study.

In order to handle multiple message exchanges on the same network connection, the TestBATN engine needs to store the connection object until the connection is requested to close by one of the messaging systems according to some messaging protocol. The engine should also know which messaging step i.e. Send Message, Receive Message, or ListenMessage is belonging to which transaction (or network connection). There is already a parameter for messaging step constructs, called transactionID, in TestBATN TDL to bind two corresponding message steps, i.e. Send-Receive, Receive-Send or Listen-Listen. However, it was optional for the single step message exchange schemes (e.g. connectionless protocols like UDP). This parameter is required in the new version to identify the correspondence between the message steps and the transactions unambiguously. In the previous version, the TestBATN engine closes the connection object (ex: TCP/IP connection) of the transaction after the response is received or sent within request-response pattern as it was the assumption. However, in DICOM messaging there may be more than one message exchanges in any order (not always in send-receive pattern) where end of transaction depends on the messaging protocol itself or some condition in the exchanged messages. In other words, the message protocol may dictate a specific message choreography between the systems over same connection like “begin-transaction - send - send - send - receive - send - end-transaction” or it may specify that end of transaction depend on the content of last message (a special message that says the other system to end the transaction). In both ways, it is protocol dependent. Therefore, the test framework should enable the test designer to explicitly set the point where the transaction will end. In the new version of TestBATN framework, the following two further messaging constructs are defined in TDL for this purpose.

- *BeginTransaction:* The construct tells the engine that the transaction will begin with the messaging steps that come after this construct. It has a parameter, transactionID, which will be the identifier for the transaction and used to match the transaction with the messaging steps. The messaging steps do not have to strictly follow the BeginTransaction, there may be other test steps between them. The construct is just a syntactic sugar that improves the readability of the test case and eases the interpretation of the test case by TestBATN engine. The first messaging step of the transaction after the BeginTransaction construct will trigger the connection initiation.

- *EndTransaction:* The construct notifies the TestBATN engine that the transaction has finished. Then the engine should close the connection and delete the connection object. All the messaging steps that belong to the transaction should occur between these tests constructs for a single transaction and will be handled over the same connection object (ex: TCP/IP connection).

5.3.3.2 Development of Necessary Adapters for DICOM testing

While some enhancements are needed in network layer, the TestBATN framework with its modular architecture is able to support DICOM testing in terms of other parts; content handling and validation and reporting by developing plug-ins specific for DICOM standard without any changes in the core framework.

First of all, there is a need for messaging adaptors that is able to send and receive DICOM messages with DICOM network communication protocol. A receiver, called DICOMReceiver and a sender, called DICOMSender is developed for this purpose. The dcm4che [73] open source library (org.dcm4che.net) has been used to implement these adapters to handle the DICOM communication protocol and message handling.

Any transport layer receiver that will operate within TestBATN should implement the ITransportLayerReceiver interface. The interface methods are as follows:

- *bindServerSocket(transactionConfig):Object:* This is the method called when the test engine is playing the message receiving side in some point in the scenario and the server socket for the connection needs to be created and bound to some port in the TestBATN server. In other words, it is called for transactions that the first message step is ReceiveMessage. This method will be called after the configuration phase before the execution of test steps in order to prevent other programs in the server to bind the port by mistake. It returns an object representing the server socket functionality according to the protocol (ex: for a simple TCP/IP, the object is java.net.ServerSocket). For the DICOMReceiver, the related parts of the codes in dcm4che are collected in this method and the method returns org.dcm4che.net.NetworkApplicationEntity which actually represents a DICOM server that provides services over the network.
- *startPortListening(serverSocket):Object:* Similarly, this method is only called for the

ReceiveMessage constructs that is a first message step of a transaction (i.e. test engine is playing a server role). The method is called when the transaction begins with the ReceiveMessage construct. It takes the server socket object (that the bindServerSocket method created) as an input and returns the connection object (ex: for a simple TCP/IP, the returned object will be java.net.Socket). For the DICOM receiver, the returned object will be org.dcm4che.net.Association that represents the DICOM association that will be established in later steps.

- *receiveMessage(transactionConfig, connection):Message:* Similarly, this method is only called for the ReceiveMessage constructs that is a first message step of a transaction to get the first message. It takes connection object (that startPortListening returns) and the network configuration and returns the received message. For the DICOMReceiver, it handles the establishment of DICOM association and then retrieving the first DICOM message over the association. The constructed message also contains the parts related with the association.

- *receiveMessageFromExistingConnection(connection):Message:* This method is called for all ReceiveMessage constructs that are not the first messaging step of a transaction. In other words, it is used to receive message from an existing connection. For the DICOMReceiver, it takes the connection object (org.dcm4che.net.Association) as input and receives further DICOM messages over an established association.

- *configure(parameter, value):* This method is used to configure the receiver in general. TestBATN message receivers define their configuration parameters in their metadata. DICOMReceiver has several configuration parameters.

- *getConnectionObject():Object:* This should return the connection object if the receiver is the first one that established the connection by the receiveMessage method.

- *closeConnection(connection):* This method should close the given connetion object and called with the EndTransaction construct for the transaction.

- *closeServerSocket(serverSocket):* This method should unbind the server socket and close it. It will be called when the test case is finished.

As seen from the interface, the developer can choose the server socket and connection objects that the receiver will return by its methods according to the target protocol. However, the objects must be consistent between the sender and receiver handlers of the protocol. The output of the receiver, that is the content of the Message object returned from the receiveMessage and receiveMessageFromExistingConnection methods, is defined in the metadata of the receiver. In this way, TesBATN engine will get the correct parts of the message from the Message object. The metadata of DICOMReceiver is given in Appendix D. As seen from the metadata; the DICOMReceiver parses and breaks the DICOM message exchanges into five pieces. For this example, this message that we perceive as a single message in terms of TesBATN abstraction is not a single DICOM message that is sent over the network at a time. It is actually a set of association establishment messages and one actual DICOM message partitioned in DICOM command and DICOM dataset parts. However, from testing perspective there is no point in dividing these message exchanges and binding each of them to a single TestBATN messaging step, since it will be wasteful to force the test designer to write three more test steps for each association establishment, i.e. one SendMessage for association request (A-ASSOCIATE-RQ), one ReceiveMessage for association accept or reject (A-ASSOCIATE-AC or A-ASSOCIATE-RJ) and one SendMessage for association release. Instead with this modeling approach, the first messaging step will handle the association message exchanges together with the first actual message.

Similar to receivers, any TestBATN transport layer sender should implement the following ITransportLayerSender interface:

- *sendMessage(message, transactionConfig):Object:* This method is called for SendMessage constructs that are the first messaging step of a transaction. In other words, it is used when the TestBATN engine is playing the client role (the side which triggers the communication first) in the communication. It takes the message parts and network configurations (ex: port and network address to send the message) as input and send the message to the specified network address after constructing the message from the given parts. The method should return the connection object established with the other side. For DICOMSender, this method establishes the association and sends the first message over the association. It returns the org.dcm4che.net.Association object as connection object.

- *sendMessageFromExistingConnection(message, existingConnection):* This method is called for all SendMessage constructs that are not the first messaging step of a transaction. For DICOMSender, it is used to send further messages from an established association.

- *forwardMessage(message, transactionConfig):Object:* The method is the same as the sendMessage method where the only difference is that it is used for ListenMessage constructs to forward the messages received from one system under test to another system under test.

- *forwardMessageFromExistingConnection(message, existingConnection):* The method is the same as the sendMessageFromExistingConnection method where the only difference is that it is used for ListenMessage constructs to forward the messages received from one system under test to another system under test.

- *configure(parameter, value):* Similar functionality with the method in the receiver interface.

- *getConnectionObject():Object:* Similar functionality with the method in the receiver interface.

- *closeConnection(connection):* Similar functionality with the method in the receiver interface.

- *closeServerSocket(serverSocket):* Similar functionality with the method in the receiver interface.

The plug-in metadata for DICOMSender is given in the Appendix D. As seen from the metadata, DICOMSender has several configuration parameters and four input message parts to construct a DICOM message that will be sent to system under test. The first input is the SOP Class UID of the message that is the type of the DICOM message (ex: CT Image Storage). The second is the transfer syntax of the message to be sent. The third input is the command (header) part of the message (header parameters for C-STORE). Finally, the last one is the actual DICOM dataset (ex: the DICOM CT image object).

In order to use DICOMSender adapter for SendMessage construct, the test designer needs to provide all of these inputs to the construct in TestBATN TDL. DICOM defines several basic services like C-STORE, C-FIND, C-GET, etc that are used in DICOM Service Class Specifications. These DIMSE services have different command structure and parameters which make it difficult for test designer to remember and generate the command object for each SendMessage construct. Therefore, another messaging adapter, DIMSESender, in packaging layer is developed for to handle all DIMSE messages easily.

In TestBATN each packaging sender should implement the following interface, IPackagingLayerSender:

- *sendMessage(message, transactionConfig):Message:* The method called for each SendMessage construct that use the adapter for packaging layer. It takes some message parts as input and construct bigger message parts that the transport layer senders can understand. The metadata of DIMSESender is given in the Appendix D. As shown in the adaptor metadata, DIMSESender takes more primitive paramaters as input and construct the four parameters that DICOMSender takes as input to send the message.
- *configure(parameter, value):* Similar functionality with the method in transport layer interfaces.

While describing how we can manage the message receiving and sending by DICOM receivers and senders, the encoding of data received or to be sent is not mentioned. Normally, when we receive a message over the network or read a file it will be in raw byte array format. During the test execution, this format can be preserved. However, then every validation adapter or messaging adapter needs to handle this raw format when they handle the data. Similar to XML content case for HL7 v3 messages, when we have validators or other mechanisms (XPATH) to process a common format, there is a need for some mechanism to transfer the messages into this format. In this case study, for DICOM, since an open source DICOM library is used to develop messaging and validation adapters, a specific object model, or as called TestBATN Data Model, is necessary to store and use the data during the test execution.

Therefore, a data model plug-in, called “dicom” is developed. Each TestBATN Data Model plug-in should implement the ITFDataHolder interface which has the following methods:

- *setDirectValue(object):* This methods is the setter for the actual data that the this model is storing inside. For “dicom” data type, the object is org.dcm4che2.data.DicomObject from the open source dcm4che library.
- *getDirectValue():Object:* Returns the actual object that is stored in this data holder.

- *clearValue():* Nullify the object that is stored in this data holder.

- *getDataType():String:* Returns the identifier for the data type. For “dicom” data type it returns “dicom”.

- *serializeToByteArray(encoding):Object:* Any TestBATN Data Model has to implement serialization and deserialization to/from byte array. The method should return the TestBATN default data model object for raw data. For “dicom” data type, the dscm4che library is used to serialize the object into a byte array with the given transfer syntax.

- *serializeToString(encoding):Object:* Any TestBATN Data Model has to implement serialization and deserialization to/from string, since this can be used in test reporting to show the message content to system under tests in some readable format. The dcm4che library has a specific text format to show DICOM objects and it is used in this method for serialization.

- *serializeToXML(encoding):Object:* TestBATN Data Models may choose to implement serialization and deserialization to/from XML format, as it will be very helpful in data processing by TestBATN expression language based on XPATH. The dcm4che library defines a specific XML format to represent DICOM objects and this is used for serialization to XML.

- *deserializeFromByteArray(value, encoding):* It is the opposite of the serialization, gets the TestBATN default byte array data model with the given encoding and constructs the own object model.

- *deserializeFromString(value, encoding):* It is the opposite of the serialization.

- *deserializeFromXML(value, encoding):* It is the opposite of the serialization.

- *isAbleToSerializeTo(dataType, encoding):Boolean:* In general, there may be serialization or deserialization between other data types. This method is used to ask the model if it has the capability to serialize to given data type with the given encoding. Since “dicom” data type does not serialize to any other data type in TestBATN it always return false for the others.

- *isAbleToDeserializeFrom(dataType, encoding):Boolean:* Similar to isAbleToSerializeTo method.

- *deserializeFrom(value, encoding):* Called for deserialization from data types other than XML, string and byte array.

- *serializeTo(value, encoding):* Similar to deserializeFrom method.

- *getEncoding():String:* Returns the current encoding of the data model.

- *setEncoding(encoding):* Set the encoding of the data model.

After defining the data model, it can be used to define the types of the input and output parameters of all types of TestBATN plug-ins; messaging adapters, validation adapters, etc. As shown in the metadata of DICOMReceiver and DICOMSender, some parameters are defined as "dicom". This means that the receiver and sender gets or gives the data in this format.

With these adapters, we can define test cases that can receive and send DICOM message and can store the message parts in "dicom" type variables. Then we need mechanisms to validate the DICOM datasets according to DICOM specification and process the data and verify element values based on DICOM and IHE SWF specifications. DICOM standard does not have an official schema format. However, there is a free tool set, DICOM Validation Toolkit that represents the restrictions in the specification for the structure of datasets, data type, multiplicity, enumerated values of DICOM elements as a special type of schema. Figure 5.6 shows a part of the schema for "Patient Module" (IOD) in "CT Image Storage" service class. In order to utilize these schemas, the DICOMValidator is developed as a TestBATN Verification Adapter.

In TestBATN, all verification adapters should implement IVerificationAdapterInterface that has the following methods:

- *verify(functionName , parameters):Boolean:* This method is called when the verification adapter is used for the VerifyContent construct to verify some data in the test scenario. The adapter can support different verification functions, and the method parameter "functionName" enables the test designer to choose one of these verification functions. For DICOMValidator, there is only one type of verification, so this parameter is optional and not important. The parameters are the input for the verification in terms of TestBATN data types. As shown in the metadata of the DICOMValidator in Appendix D, the DICOMValidator takes four parameters as input; SOP Class UID that is the class of DICOM dataset to be validated against (ex: CT Image Storage), DIMSE command name, command part of DICOM message and dataset part of DICOM message. The method should return the Boolean value that indicates the result of verification.

- *getVerificationResult():EvalAdapterResult:* The textual report of verification is retrieved later by this method after the verification.

Validating against a schema is not enough for scenario based testing of IHE SWF workflows and further DICOM restrictions that needs comparison of some DICOM element values with scenario parameters or other DICOM elements. However, in our case we do not need special mechanisms since the “dicom” data type has the ability to serialize/deserialize to/from XML. The test designer can assign the variable in “dicom” type to a variable in TestBATN XML type and then write XPATH expressions on this special format XML to access individual DICOM elements.

5.3.4 A Sample Test Case: IHE Connect-a-thon SWF-OF-Modality Test Scenario

One of the test cases in the TestBATN IHE SWF test suite is given in the Appendix D as a sample. This test case is the implementation of IHE SWF-OF-Modality Connect-a-thon test scenario where the workflow and definition of the scenario is show in Figure 5.7 Although, the scenario is an interoperability test scenario, since this case study is used to prepare DSS/Order Filler and Image Manager actor implementations to the connect-a-thon, the scenario is implemented as a conformance test case that tests the DSS/Order Filler implementations. In this section, some parts from the test case definition is illustrated to show how test constructs and the adapters developed for DICOM testing is used in this case study.

The Figure 5.8 shows the definition of actors and the configuration part. Two actors are defined in the test case as this will be a conformance test case. Since it will test DSS/Order Filler implementations, one of the actor will be named as DSSOF. The other actor is the TestBATN engine that simulates all the ADT, Order Placer(OP), Modality and ImageManager actors in the scenario. The CPA definition for the RAD5-Query Modality Worklist interaction between the Modality and DSSOF is shown in the figure.

```
<!--Parties Participating the Test-->
<ActorUnderTest>ADT_OP_MODALITY_IM</ActorUnderTest>
<!-- The system under test is playing the DSS/Order Filler role -->
<ActorUnderTest>DSSOF</ActorUnderTest>
<!--TestBATN is playing Patient Registration (ADT), Order Placer(OP), Image Manager(IM)
and Modality (imaging device) roles-->
<TestCaseCoordinator>ADT_OP_MODALITY_IM</TestCaseCoordinator>
<ConfigurationGroup>
   ...
   <CPA id="DICOM_INT1" fromActor="ADT_OP_MODALITY_IM" toActor="DSSOF">
      <CPAType>Interaction for RAD5 Transaction between Modality-DSSOF</CPAType>
      <CPAId>http://www.srdc.com.tr/cpa3</CPAId>
   </CPA>
   ...
</ConfigurationGroup>
```

Figure 5.8: Sample Test Case Configuration Definition

```
<BeginTransaction tid="RAD5" cpaID="DICOM_INT1"/>
<Assign id="ASGN_4" variableRef="cfind_dataset_xml" append="no_append">
   <Expression>#testsuites/Dicom_IHEPIR/msgtemplates/SWF_OF_Modality_Interface/mwl-Case2.xml</Expression>
</Assign>
<Assign id="ASGN_5" variableRef="cfind_dataset_dcm" append="no_append">
   <Expression>$cfind_dataset_xml</Expression>
</Assign>
<SendMessage id="S06" description="The Modality will send the RAD5-Modality Worklist Query(DICOM C-FIND) to your
 application" transactionID="RAD5" toActor="DSSOF">
   <Transport transportHandler="DICOMSender">
      <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
      <ConfigurationAttribute name="presentationContextID">1.2.840.10008.5.1.4.31</ConfigurationAttribute>
      <ConfigurationAttribute name="remoteAETitle">DCM_SERVER</ConfigurationAttribute>
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
   </Transport>
   <Packaging packagingHandler="DIMSESender">
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
   </Packaging>
   <SetMessage description="C-FIND REQUEST">
      <Content><VariableRef>dimse-cmd-req</VariableRef></Content>
      <Content><VariableRef>cfind_dataset_dcm</VariableRef></Content>
      <Content><VariableRef>msgId</VariableRef></Content>
      <Content><VariableRef>cuid</VariableRef></Content>
      <Content><VariableRef>priority</VariableRef></Content>
   </SetMessage>
</SendMessage>
<ThreadGroupRef nameRef = "TG02"/>
```

Figure 5.9: A Sample Messaging Step for DICOM Messaging

The Figure 5.9 shows the part of the test case where the Modality sends the RAD5-Modality Worklist (MWL) Query to the system under test (DSS/OrderFiller). The BeginTransaction construct states that transaction will start and will be based on the network configurations done for the interaction “DICOM_INT1”. Since the TestBATN engine is playing the Modality role, it should construct the DICOM MWL query and send it. In order to construct the message, we use a sample message, “mwlCase2.xml” stored in the file system together with the test case definition as message template as shown in Figure 5.10. As shown in the message template, the query is only based on patien’s name which gets the value from a placeholder variable called patientName. The first Assign construct reads the content of the file, replaces the placaeholders with their values and assign it to the TestBATN XML type variable “cfind_dataset_xml”. The second Assign construct, assign the value of this variable to “dicom” type variable “cfind_dataset_dcm”. This is the place where deserialization method of

```
<dicom>
        <!--Imaging Service Request-->
<attr tag="00080050" vr="SH" len="0"/>
<attr tag="00080090" vr="PN" len="0"/>
<!--Visit Identification-->
<attr tag="00380010" vr="LO" len="0"/>
<attr tag="00380300" vr="LO" len="0"/>
<attr tag="00081120" vr="SQ" len="-1">
<item off="102" len="-1">
<attr tag="00081150" vr="UI" len="-1"/>
<attr tag="00081155" vr="UI" len="-1"/>
</item>
</attr>
<!--Patient Identification-->
<attr tag="00100010" vr="PN" len="0">%$patientName%</attr>
<attr tag="00100020" vr="LO" len="0"></attr>
<attr tag="00100030" vr="DA" len="0"/>
<attr tag="00100040" vr="CS" len="0"/>
</dicom>
```

Figure 5.10: A Sample Message Template for DICOM Tests

DICOM Data Model adapter is called for transforming the content in special DICOM XML format to real DICOM object model.

The SendMessage construct states that the message will be sent to DSSOF actor (toActor parameter) for the transaction RAD5. It also states that transport will be handled by the DICOMSender adapter with the given configuration parameters. For example, the value of “presentationContextID” (1.2.840.10008.5.1.4.31) is the UID of the SOP Class “Modality Worklist Information Model - FIND” which is the class for the DICOM MWL Query. In this way, DICOMSender uses this attribute in the association negotiation. For the packaging, the consturct states that the DIMSESender adapter will be used. Since there is a packaging handler and it is DIMSESender to be used, the inputs given to the construct must match with the DIMSESender adapter inputs. Each Content tag gives the message parts in order as stated in the metadata of DIMSESender. The first input is the name of command that is taken from the String type variable “dimse-cmd-req” which is C-FIND-RQ. The second input is the DICOM dataset for the query that we construct with Assigns. Finally, other inputs are other required parameters for C-FIND command. When the SendMessage contruct is executed by the TestBATN engine, first the sendMessage method of DIMSESender is called. The method gets the inputs and prepare message parts that DICOMSender can understand. Then DICOMSender is configured with the parameters and its sendMessage method is called since it is the first messaging step of the transaction. The DICOMSender setup the association and send the message that it constructs from the parts. The last line in the Figure 5.9 redirects the execution to another ThreadGroup with ID “TG02”.

```
<ThreadGroup name="TG02" description="Loop for C-FIND responses">
   <Thread name = "TH02" description="Loop for C-FIND responses">
      <ReceiveMessage id="R4" transactionID="RAD5" fromActor="DSSOF" description = "Please send the responses
for the RAD5 Modality Worklist Query (C-FIND RSP)">
 <Transport transportHandler="DICOMReceiver">
    <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
    <ConfigurationAttribute name="presentationContextID">1.2.840.10008.5.1.4.31</ConfigurationAttribute>
    <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
    <ConfigurationAttribute name="releaseWhenFinished">false</ConfigurationAttribute>
 </Transport>
 <VariableRef>assoc_rq</VariableRef>
 <VariableRef>assoc_ac</VariableRef>
 <VariableRef>pdv_command</VariableRef>
 <VariableRef>pdv_dataset</VariableRef>
 <VariableRef>a_abort</VariableRef>
      </ReceiveMessage>
      <Assign id="ASGN_1" variableRef="cfind_rsp_xml" append="no_append">
 <Expression>$pdv_command</Expression>
      </Assign>
      <TestAssertion id="TA4" description = "Check if status is pending">
 <VerifyContent verificationAdapter="xpathadapter"
    description="Check the CFIND status">
    <Expression>$cfind_rsp_xml</Expression>
    <Expression>/dicom/attr[@tag='00000900']/text() = '65280' or /dicom/attr[@tag='00000900']/text() =
'65281'</Expression>
 </VerifyContent>
 <WhenTrue>
    <ThreadGroupRef nameRef = "TG03"/>
 </WhenTrue>
      </TestAssertion>
   </Thread>
</ThreadGroup>
```

Figure 5.11: Sample Test Case Fragment for a DICOM Transaction

The Figure 5.11 shows the ThreadGroup definition that will receive the responses to the query in a loop. DICOM C-FIND service protocol states that the query responser should return each matching result by a C-FIND-RSP message with a value of ’PENDING’(65280 or 65281) in the DICOM Command Status tag(00000900) which indicates the operation is “PENDING”. When all the matching results are return, protocol statest that the responder sends a last C-FIND-RSP message with the status tag value “0” as “SUCCESS”. In order to apply the protocol, ReceiveMessage construct gets the message by using the DICOMReceiver and stores the DICOM command part in pdv_command variable and DICOM dataset part in pdv_dataset variable. The next Assign construct serializes the DICOM command part into the XML format in cfind_rsp_xml variable. The TestAssertion construct check the value of the status attribute by an XPATH expression and when the result is “PENDING” it states that the execution will go to ThreadGroup TG03. If it is not the “PENDING” but the “SUCCESS”, the execution of ThreadGroup is finished and the execution returns back to the main part after the call of ThreadGroup TG02.

The Figure 5.12 shows the definition of ThreadGroup TG03. The first Assign construct use a function from default function library to accumalate number of query matches on the variable “numofmatch”. This will later be checked if it is correct for the scenario. The other assigns

```
<ThreadGroup name="TG03" description="Validation of C-FIND responses">
   <Thread name = "TH03" description="Validation of C-FIND responses">
      <Assign id="ASGN_2" variableRef="numofmatch" append="no_append">
 <Expression namespaceBindings="{srdc = http://srdc.testframework.default}">
    srdc:add($numofmatch, '1')</Expression>
      </Assign>
      <Assign id="ASGN_3" variableRef="cfind_rsp_xml_dataset" append="no_append">
 <Expression>$pdv_dataset</Expression>
      </Assign>
      <Assign id="ASGN_3a" variableRef="studyInstanceUID" append="no_append">
 <Expression>$cfind_rsp_xml_dataset/dicom/attr[@tag='0020000D']/text()</Expression>
      </Assign>
      ...
      <TestAssertion id="TA5" description = "Validate DICOM Content according to Table F.7.2-1">
 <VerifyContent verificationAdapter="dicomadapter"
    description="Validate DICOM Content">
    <Expression>$cuid</Expression>
    <Expression>$dimse-cmd</Expression>
    <Expression>$pdv_command</Expression>
    <Expression>$pdv_dataset</Expression>
 </VerifyContent>
 <VerifyContent verificationAdapter="xpathadapter"
    description="Check whether Image Manager returns response message id properly.">
    <Expression>$cfind_rsp_xml</Expression>
    <Expression>/dicom/attr[@tag='00000120']/text() = '1387'</Expression>
 </VerifyContent>
      </TestAssertion>
      <ThreadGroupRef nameRef = "TG02"/>
   </Thread>
</ThreadGroup>
```

Figure 5.12: Sample Valitation Descriptions for DICOM Testing

gets some parameter values to use them in later parts of the scenario. The first VerifyContent construct use the DICOMValidator to validate the received response against the schema that is prepared for DICOM C-FIND-RSP messages. The second VerifyContent checks if the message ID that is used in query is returned correctly in the query response. Finally, the execution is redirected to ThreadGroup TG02 again, which now runs as a GOTO construct and finalize the loop. This loop is executed for each query match until the last C-FIND-RSP with the status “SUCCESS” as the protocol dictates.

The VerifyContent construct in the Figure 5.12 for the basic syntactic validation of DICOM C-FIND-RSP message utilize the DICOMValidator as the verification adapter. The “cuid” variable gives the SOP Class UID which is Modality Worklist Information Model - FIND (1.2.840.10008.5.1.4.31) to notify the verifier that the validation should be done according to the schema for this SOP Class.

CHAPTER 6

CONCLUSION

In this section, the capabilities and features of the TestBATN framework will be summarized from different perspectives: the approach and the proposed abilities for testing; the facilities for test execution; and the facilities for test design and maintenance.

From the generic perspective, the TestBATN framework provides the following the testing abilities:

- Integrated Testing: Partly automating a test process that is, providing different tools for different layers of the interoperability stack and doing the rest manually results in human labor intensive, error prone, costly to develop test processes. TestBATN framework provides an integrated testing environment and a methodology to combine all testing features and phases for different layers; business process layer, document layer and communication layer into a single test scenario.
- Scenario Based Testing: Basic conformance and interoperability testing does not ensure that the information in the transmissions accurately represents the intentional semantics of the applications. TestBATN framework enables scenario based testing where SUTs are given a test scenario and a set of scenario requirements and are requested to operate their systems accordingly.
- Web-based Testing: TestBATN framework enables vendors to test their products over Web anytime, anywhere and with any party willing to do so. Therefore, it can be both used as a Web based test platform for vendors to test their systems conformity and interoperability with other system or as a test platform for interoperability testing events.

- Unrestricted Support for Conformance and Interoperability Testing of B2B Standards: The TestBATN framework provides a modular approach where different modules supporting different communication protocols, data formats, and testing tools or functionalities can be plugged in to the framework.

- Automated Preliminary Preparation Phases: Automation of testing implies non-interference with the system in its native state of execution. The TestBATN framework provides a holistic approach by involving configuration management and other preliminary preparation processes into the testing process.

- Support for Complex Test Scenarios: The TestBATN framework provides the necessary instructions and the execution model to support complex test scenarios that require concurrent execution flows, alternative test scenario paths and the repetitive steps.

- Conformance Validation for Communication and High Level Messaging Protocols: The TestBATN framework with its messaging interfaces and pluggable messaging adaptors provides the ability to test the conformance of the communication performed according to various different communication and messaging protocols as specified in the base standards. The approach also enables testing different part of the messages.

- Reusability of Existing Testing Materials: Some standards or interoperability initiatives publish integration profiles together with supplementary materials like XML schemas of sample messages, schematrons for business rules, and the code lists related to the standard or profile. These materials can easily be used for conformance and interoperability testing purposes if they can be integrated into the testing platform. The TestBATN framework enables test designers to use these already available materials by its Test Assertion Interfaces and modular approach.

- Support for Non-XML Content: The TestBATN framework enables conformance and interoperability testing of non-XML messages by its abstract representation model with user-defined object models and with the adaption of its expression language to devise tests on these models.

- Dynamic Test Scenarios: The TestBATN framework enables the utilization of random data providers in a test scenario and combines this ability with scenario based testing to make the design of dynamic test scenarios possible. For each instance of dynamic test scenarios, the Human Test Drivers confront with different scenario requirements which make the testing more strict and strong. This ability of the TestBATN framework can be very important for the certification programs.

- Integrate the Manual Testing into the Automation: Some standards or profiles have conformance criteria which cannot be tested without manual intervention or interaction with the Human Test Driver of the SUT. The TestBATN framework has a specific instruction to integrate the manual testing or user interactions with the help of TestBATN Control and Monitoring GUI.

From the vendors' perspective, that is, the users who will benefit from the automated conformance and interoperability testing in terms of finding the nonconforming or non-interoperable parts or errors in their systems, the TestBATN framework provides the following features and benefits:

- Test Scenario Presentation: The TestBATN framework defines a presentation model to present a test case definition to the Human Test Drivers, the users who are operating the SUTs during testing process. This model facilitates clear presentation of the test scenarios in different formats like a sequential listed test steps, graphical presentation with actor or sequence diagrams. In this way, the vendors will have a better understanding of the scope and the methodology of the test scenario.

- Test Step Notion: The TestBATN framework adopts a test step notion which helps the Human Test Driver to understand and monitor the test execution better. With this notion, the Human Test Driver is also able to observe the exact point of errors or nonconformity of its system.

- Run-Time Monitoring and Control Ability: The TestBATN framework with its binding between the test scenario presentation model and the test execution, enables run-time monitoring for the test execution. A user can monitor the reports and verdicts for executed test steps and the step that the engine is executing at that time. Therefore, the Human Test Drivers easily relate their own actions with the executed steps and easily follow the instructions in the test scenario.

- Detailed and Structured Test Reports: In the TestBATN framework, a structured report is generated for each test step. Since the adaptor implementers are free to use any format they choose and determine the scope for the test reports, more detailed reporting is possible in the framework.

Finally, for the responsible organizations (i.e. standard organizations, interoperability initiatives, organizations that publish integration or implementation specifications) that need conformance and interoperability testing/or certification platform/or program, the TestBATN framework provides the following features:

- High Reusability of Modules and Test Case Definitions: The modular architecture of the TestBATN framework facilitates the reusability of specific modules, adaptors, object data models, function libraries, and the parts of test case definitions (i.e. Thread definitions) for defining the test case for different standards, and even different domains.

- Easy Maintenance: Standard development is a continuous process and a new version may be published each year with some additions or updates on several parts of the standard. Furthermore, for certification or annual test events, test cases need to be updated to prevent possible specific adjustments in the SuTs to pass the tests with some specific requirements. The structure of the TestBATN TDL and by its parameterization ability (by using the variables), the test scenarios can easily be updated and maintained continuously.

- Easy Design for Test Scenarios: The TestBATN framework provides a graphical environment where the test designers can assemble the reusable test constructs and pluggable adaptors for conformance and interoperability test case definitions.

- Better Understanding on Test Case Definitions: In the state-of-the-art methodologies or testbeds, the scenario design and implementation requires a deep knowledge of the testbed functionalities and internals. Therefore, highly experienced technical people can develop test scenarios. In other words, for a specific standard or domain, the responsible person needs to learn the details of the standard and understand its conformance criteria in order to design and implement to-the-purpose test scenarios. The TestBATN framework, with its graphical tools, aims at providing a collaborative environment for technical persons and standard experts to get actively involved in the test scenario design and implementation.

Although TestBATN framework provides wide range of abilities and functionalities for the automated conformance and interoperability testing, there is still a significant amount of work for organizations that wants to design and maintain conformance and interoperability test scenarios for a certification like procedure for vendors. In general, standards have many lines of textual specifications that need to be analyzed and covered by some test script in a conformance and interoperability test scenario. This makes it very hard to design, organize and maintain your test cases for a complete standard regardless of facilities that your test framework or the test description language provides. Therefore, there is a need for automated or semi-automated mechanisms that can generate test case descriptions based on computable resources, i.e. Schemas, Business Process Definitions, Rules, etc that describes the specifications in the target standard. Many standard development organizations already provide such resources as a normative part of the standard. As a future work, TestBATN may be extended with such tools and mechanisms to automatically generate TestBATN test case descriptions from the computable resources provided together with the standard. Furthermore, the conformance statement of vendors to declare how they conform to standards may also be defined in a structured way so that it can be input to test case generation process and specialized test cases can be generated. In general standards have optional parts and the vendors declare conformance to these parts in their conformance statement if they have implemented it. In this way, anything stated in the conformance statement will be tested and it will ease the organization of test cases for optional parts of the standard.

Another future work for TestBATN is to transfer the platform architecture on OSGi platform so that the dynamicity and modular architecture can be used more effectively. In this way, the TestBATN framework can be used as a remote platform where any organization can develop further adapters, test suites, test case definitions, deploy them into the TestBATN and maintain them remotely over Web.

REFERENCES

[1] IEEE Dictionary, Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries, New York, 1990.

[2] Brown and Reynolds, "Strategy for production and maintenance of standards for interoperability within and between service departments and other healthcare domains", CEN/TC 251 Health Informatics, CEN/TC 251/N00-047.

[3] hl7.org [Internet], Health Level 7 [cited 2010 Feb 25]. Available from: http://www.hl7.org/

[4] HL7 Version 3 Standard: Transport Specification - ebXML, Release 2. HL7 [Internet]. December 2009 [cited 2010 Feb 25]. Available from: http://www.hl7.org/v3ballot/html/infrastructure/transport/transport-ebxml.htm

[5] HL7 Version 3 Standard: Transport Specification - Web Services Profile, Release 2. HL7 [Internet]. December 2009 [cited 2010 Feb 25]. Available from: http://www.hl7.org/v3ballot/html/infrastructure/transport/transport-wsprofiles.htm

[6] Transport Specification: MLLP, Release 2. HL7 [Internet]. December 2009 [cited 2010 Feb 25]. Available from: http://www.hl7.org/v3ballot/html/infrastructure/transport/transport-mllp.htm

[7] HL7 Clinical Document Architecture, Release 2. HL7 [Internet]. August 2004 [cited 2010 Feb 25]. Available from: http://hl7.org/library/Committees/structure/CDA.ReleaseTwo.CommitteeBallot03.Aug.2004.zip

[8] The Systematized Nomenclature of Medicine (SNOMED). International Health Terminology Standards Development Organization [Internet]. February 2010 [cited 2010 Feb 25]. Available from: http://-www.ihtsdo.org/snomed-ct/.

[9] International Statistical Classification of Diseases and Related Health Problems, 10th Revision (ICD-10), Second Edition. World Health Organization [Internet]. 2007 [cited 2010 Feb 25]. Available from: http://www.who.int/whosis/icd10/.

[10] Logical Observation Identifiers Names and Codes (LOINC) [Internet]. July 2009. [cited 2010 Feb 25]. Available from: http://www.regenstrief.org/loinc/

[11] Northern European Subset, NES [Internet]. Available from: http://www.nesubl.eu/ [last visited date 25/05/2011]

[12] ebXML Business Process Specification Schema. OASIS [Internet]. May 2001 [cited 2010 Feb 25]. Available from: http://www.ebxml.org/specs/ebBPSS.pdf

[13] HL7 v3 Ballot Site. HL7 [Internet]. May 2009 [cited 2009 October 28]. Available from: http://www.hl7.org/v3ballot2009may/html/welcome/introduction/index.htm

[14] ihe.net [Internet]. Integrating the Healthcare Enterprise (IHE) [cited 2010 Feb 25]. Available from: http://www.ihe.net/.

[15] López D, Blobel B. Enhanced semantic interoperability by profiling health informatics standards. Methods Of Information In Medicine [serial on the Internet]. (2009), [cited 2010 February 25]; 48(2): 170-177.

[16] hitsp.org [Internet]. Healthcare Information Technology Standards - HITSP; c2008-2009. [cited 2010 Feb 25]. Available from: http://www.hitsp.org/

[17] sagliknet.gov.tr [Internet]. Saglik Net; c2008-2009. [cited 2010 Feb 25]. Available from: http://www.sagliknet.saglik.gov.tr

[18] Event-driven Test Scripting Language (eTSL), OASIS ebXML Implementation Interoperability and Conformance (IIC) TC, Working Draft 0.82, November 2007, http://www.oasis-open.org/committees/download.php/22445/eTSL-draft-082.pdf [last visited date 25/05/2011]

[19] NIST Manufacturing Business to Business (B2B) Interoperability Testbed, http://www.mel.nist.gov/msid/b2btestbed/

[20] Korean B2B/A2A Interoperability Testbed (KorBIT), http://www.torpedo.co.kr/eng/eng_success_03.htm [last visited date 25/05/2011]

[21] Web Services Interoperability Organization - WS-I test tools, http://www.ws-i.org/deliverables/workinggroup.aspx?wg=testingtools [last visited date 25/05/2011]

[22] ETSI TTCN-3 Testing and Test Control Notation, http://www.ttcn-3.org/ [last visited date 25/05/2011]

[23] European Telecommunication Standard Institute (ETSI), http://www.etsi.org/ [last visited date 25/05/2011]

[24] Worldwide Interoperability for Microwave Access (WiMax), http://www.wimaxforum.org/ [last visited date 25/05/2011]

[25] IP Multimedia Subsytem (IMS), http://www.3gpp.org/ftp/Specs/html-info/23228.htm [last visited date 25/05/2011]

[26] HL7 Message Maker, http://www.itl.nist.gov/div897/ctg/messagemaker/ [last visited date 25/05/2011]

[27] TestBATN (TM) - Testing Business, Application, Transport and Network Layers, http://www.srdc.com.tr/index.php?option=com_content&task=view&id=182&Itemid=192 [last visited date 25/05/2011]

[28] Global eBusiness interoperability test bed methodologies, GITB, http://www.ebusiness-testbed.eu/publicaties/4762 [last visited date 25/05/2011]

[29] CEN Workshop Aggrement 16093:2010 - Feasibility Study for a Global eBusiness Interoperability TestBed (GITB), ftp://ftp.cen.eu/CEN/Sectors/TCandWorkshops/Workshops/CWA16093TestBed.pdf [last visited date 25/05/2011]

[30] Ensuring Interoperability with Automated Interoperability Testing, http://www.etsi.org/WebSite/document/aboutETSI/CTI/-Ensuring%20Interoperability%20with%20Automated%20Interoperability%20Testing_rev6a.pdf [last visited date 25/05/2011]

[31] Namli T., Aluc G., Dogac A. An Interoperability Test Framework for HL7 based Systems, IEEE Transactions on Information Technology in Biomedicine Vol.13, No.3, May 2009, pp. 389-399.

[32] Namli T., Dogac A., Sinaci A. A., Aluc G., Testing the Interoperability and Conformance of UBL/NES based Applications eChallenges Conference, October 2009, Istanbul, Turkey. http://www.srdc.metu.edu.tr/webpage/publications/2009/-NAMLI.InteroperabilityTestingUBLNES.doc [last visited date 25/05/2011]

[33] Namli T., Aluc G., Sinaci A., Kose I., Akpinar N., Gurel M., Arslan Y., Ozer H., Yurt N., Kirici S., Sabur E., Ozcam A., Dogac A. 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. http://www.srdc.com.tr/webpage/publications/2008/Final.Testing.ihic08-formatted.pdf [last visited date 25/05/2011]

[34] Namli, T., Dogac, A. Testing Conformance and Interoperability of eHealth Applications Methods of Information in Medicine, Vol. 49, No.3, May 2010, pp. 281-289. [last visited date 25/05/2011]

[35] National Health Information Network (NHIN), http://www.hhs.gov/healthit/-healthnetwork/background/ [last visited date 25/05/2011]

[36] Canada Health Infoway. http://www.infoway-inforoute.ca/ [last visited date 25/05/2011]

[37] National Health System (NHS) UK. http://www.nhs.uk/ [last visited date 25/05/2011]

[38] Dossier Médical Personnel (DMP). http://www.d-m-p.org/ [last visited date 25/05/2011]

[39] Kabak Y., Dogac A., Kose I., Akpinar N., Gurel M., Arslan Y., Ozer H., Yurt N., Ozcam A., Kirici S., Yuksel M., Sabur E. The Use of HL7 CDA in the National Health Information System (NHIS) of Turkey. 9th International HL7 Interoperability Conference (IHIC) 2008, Crete, Greece, October 2008

[40] HL7 v3 ballot site, Universal Domains, Medical Records Domain, Document Query Topic, National Health Data Repository Storyboard,http://www.hl7.org/v3ballot/html/welcome/environment/index.htm [last visited date 25/05/2011]

[41] HL7 v3 ballot site, Universal Domains, Medical Records Domain, Document Consent Topic, Data Consent Storyboard, http://www.hl7.org/v3ballot/html/welcome/environment/index.htm [last visited date 25/05/2011]

[42] HL7 v3 ballot site, Universal Domains, Medical Records Domain, Data Consent Topic, Application Roles, Consent Placer, http://www.hl7.org/v3ballot/html/welcome/environment/index.htm [last visited date 25/05/2011]

[43] HL7 v3 ballot site, Universal Domains, Medical Records Domain, Document Query Topic, Application Roles, Content Optional Document Management System, http://www.hl7.org/v3ballot/html/welcome/environment/index.htm [last visited date 25/05/2011]

[44] HL7 v3 ballot site, Universal Domains, Medical Records Domain, Document Query Topic, Application Roles, Clinical Document Directory,http://www.hl7.org/v3ballot/html/welcome/environment/index.htm [last visited date 25/05/2011]

[45] HL7 v3 ballot site, Universal Domains, Medical Records Domain, Document Query Topic, Application Roles, Content Required Document Management System, http://www.hl7.org/v3ballot/html/welcome/environment/index.htm [last visited date 25/05/2011]

[46] HL7 v3 ballot site, Universal Domains, Medical Records Domain, Data Consent Topic, Application Roles, Consent Manager, http://www.hl7.org/v3ballot/html/welcome/environment/index.htm [last visited date 25/05/2011]

[47] HL7 v3 ballot site, Universal Domains, Medical Records Domain, Document Query Topic, Interactions, Original Document With Content, http://www.hl7.org/v3ballot/html/welcome/environment/index.htm [last visited date 25/05/2011]

[48] HL7 v3 ballot site, Universal Domains, Medical Records Domain, Document Query Topic, Interactions, Find Document Metadata Query, http://www.hl7.org/v3ballot/html/welcome/environment/index.htm [last visited date 25/05/2011]

[49] HL7 v3 ballot site, Universal Domains, Medical Records Domain, Document Query Topic, Interactions, Find Document Metadata and Content Query, http://www.hl7.org/v3ballot/html/welcome/environment/index.htm [last visited date 25/05/2011]

[50] CEN/ISSS WS/BII Conformance and interoperability testing v03, Draft, for review by CEN/ISSS BII/WG3

[51] Universal Business Language, http://docs.oasis-open.org/ubl/cs-UBL-2.0/UBL-2.0.html [last visited date 25/05/2011]

[52] Northern European Subset, http://www.nesubl.eu/ [last visited date 25/05/2011]

[53] PEPPOL, http://www.peppol.eu/workpackages/about-peppol [last visited date 25/05/2011]

[54] Y. Kabak, A. Dogac, I. Kose, N. Akpinar, M. Gurel, Y. Arslan, H. Ozer, N. Yurt, A. Ozcam, S. Kirici, E. Sabur, The Use of HL7 CDA in the National Health Information System (NHIS) of Turkey, IHIC Conference, 2009

[55] Ulusal Saglik Bilgi Sistemi/Saglik-NET Entegrasyonu Ile ilgili Genelge 2008/18, http://www.saglik.gov.tr/TR/-MevzuatGoster.aspx?F6E10F8892433CFF1A9547B61DAFFE2A56515916B329A1F1 [last visited date 25/05/2011]

[56] HL7 Web Services Profile [Online], http://www.hl7.org/v3ballot2008jan/html/infrastructure/transport/transport-wsprofiles.htm [last visited date 25/05/2011]

[57] WS-Security Core Specifications [Online], http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf [last visited date 25/05/2011]

[58] HL7 The Clinical Document Architecture Release 2.0, http://www.hl7.org/v3ballot/html/infrastructure/cda/cda.htm [last visited date 25/05/2011]

[59] Minimum Health Data Sets [Online], http://www.sagliknet.saglik.gov.tr/-portal_pages/notlogin/bilisimciler/docs/msvs_ semalari.rar [last visited date 25/05/2011]

[60] National Health Data Dictionary [Online], http://www.sagliknet.saglik.gov.tr/-portal_pages/notlogin/bilisimciler/docs/usvs_ sozluk_ 1.1.rar [last visited date 25/05/2011]

[61] The Health Coding Reference Server (HCRS) [Online], http://sbu.saglik.gov.tr/SKRS2%5FListesi/. [last visited date 25/05/2011]

[62] International Classification of Diseases (ICD 10) [Online], http://www.who.int/classifications/icd/en/ [last visited date 25/05/2011]

[63] MERNIS, Central Demographics Management System[Online], http://www.nvi.gov.tr/Hakkimizda/Projeler,Spot_Mernis.html [last visited date 25/05/2011]

[64] IHE Cross Enterprise Document Sharing (XDS) Profile, ITI Technical Framework 5.0, Volume 1. IHE [Internet]. December 2008 [cited 2010 Feb 25]. Available from: http://www.ihe.net/Technical_Framework/upload/IHE_ITI_TF_5-0_Vol1_FT_2008-12-12.pdf

[65] IHE Cross-Enterprise Sharing of Medical Summaries (XDS-MS) Integration Profile, Patient Care Coordination Technical Framework 4.0. IHE [Internet]. October 2008 [cited 2010 Feb 25]. Available from: http://www.ihe.net/Technical_Framework/upload/IHE_PCC_TF_4-0_Vol_1_2008-10-10.pdf

[66] IHE Patient Identifier Cross Referencing (PIX) Integration Profile, ITI Technical Framework 5.0, Volume 1. IHE [Internet]. December 2008 [cited 2010 Feb 25]. Available from: http://www.ihe.net/Technical_Framework/upload/IHE_ITI_TF_5-0_Vol1_FT_2008-12-12.pdf

[67] medical.nema.org [Internet]. Digital Imaging and Communications in Medicine (DICOM). [cited 2010 Feb 25]. Available from: http://medical.nema.org/

[68] 32. CEN EN 13606-1, Health informatics - Electronic health record communication - Part 1: Reference model, European Committee for Standardization, Brussels, Belgium, Tech. Rep. EN 13606-1, 2007.

[69] IHE Scheduled Workflow Profile (SWF), http://www.ihe.net/Technical_Framework-/upload/ihe_tf_rev8.pdf [last visited date 25/05/2011]

[70] National Electrical Manufacturers Association (NEMA). http://www.nema.org/ [last visited date 25/05/2011]

[71] OSI Association Control Service Element (ACSE) - ISO 8649, http://www.doc.ua.pt/arch/itu/rec/product/X.htm [last visited date 25/05/2011]

[72] ISO/IEC 8822:1994 Open Systems Interconnection (OSI) Presentation service definition, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=24365 [last visited date 25/05/2011]

[73] dcm4che - Open Source Clinical Image and Object Management, http://www.dcm4che.org/ [last visited date 25/05/2011]

APPENDIX A

TESTBATN FRAMEWORK XML SCHEMAS

A.1 TestBATN Test Description Language XML Schema

```
<xsd:schema
   targetNamespace="model.testsuite.testframework.srdc.com"
   attributeFormDefault="unqualified"
   elementFormDefault="qualified"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="model.testsuite.testframework.srdc.com">
   <xsd:element name="TestSuite">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:MetaData" />
            <xsd:element minOccurs="0" ref="tns:Variables"/>
            <xsd:element maxOccurs="unbounded" ref="tns:TestCaseRefID" />
         </xsd:sequence>
         <xsd:attribute name="id" type="xsd:ID" use="required" />
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="MetaData">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:Title" />
            <xsd:element ref="tns:Version" />
            <xsd:element ref="tns:Maintainer" />
            <xsd:element ref="tns:Location" />
            <xsd:element ref="tns:PublishDate" />
            <xsd:element ref="tns:Status" />
            <xsd:element ref="tns:Description" />
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
```

```
<xsd:element name="TestCase">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element minOccurs="0" ref="tns:MetaData"/>
         <xsd:element minOccurs="1" maxOccurs="unbounded" ref="tns:ActorUnderTest" />
         <xsd:element ref="tns:TestCaseCoordinator" />
         <xsd:element minOccurs="0" ref="tns:ConfigurationGroup" />
         <xsd:element minOccurs="0" ref="tns:Variables" />
         <xsd:choice maxOccurs="unbounded">
            <xsd:element ref="tns:ThreadGroup" />
            <xsd:element ref="tns:Thread" />
         </xsd:choice>
         <xsd:element ref="tns:ThreadGroupRef" />
         <xsd:element minOccurs="0" ref="tns:TestCaseRefID" />
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="required" />
      <xsd:attribute name="description" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="javaScriptURI" type="tns:non_empty_string" use="optional"/>
      <xsd:attribute name="charset" type="tns:non_empty_string" use="optional"/>
   </xsd:complexType>
</xsd:element>

<xsd:element name="ConfigurationGroup">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:CPA" minOccurs="0" maxOccurs="unbounded" />
         <xsd:element ref="tns:PreliminaryData" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="CPA">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:CPAType" />
         <xsd:element ref="tns:CPAId"  />
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="required" />
      <xsd:attribute name="fromActor" type="xsd:IDREF" use="required" />
      <xsd:attribute name="toActor" type="xsd:IDREF" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="Variables">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:Variable" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="ThreadGroup">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:Thread" minOccurs="1" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:ID" use="required" />
      <xsd:attribute name="description" type="tns:non_empty_string" use="optional" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="Thread">
   <xsd:complexType>
      <xsd:choice maxOccurs="unbounded" minOccurs="0">
         <xsd:element ref="tns:SendMessage" />
         <xsd:element ref="tns:ReceiveMessage" />
         <xsd:element ref="tns:ListenMessage" />
         <xsd:element ref="tns:TestAssertion" />
         <xsd:element ref="tns:Assign" />
         <xsd:element ref="tns:ThreadGroupRef" />
         <xsd:element ref="tns:AskTestData" />
         <xsd:element ref="tns:BeginTransaction"/>
         <xsd:element ref="tns:EndTransaction"/>
         <xsd:element ref="tns:DecisionPoint"/>
      </xsd:choice>
      <xsd:attribute name="name" type="xsd:ID" use="required" />
      <xsd:attribute name="description" type="tns:non_empty_string" use="optional" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="ThreadGroupRef">
   <xsd:complexType>
      <xsd:attribute name="nameRef" type="xsd:IDREF" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="BeginTransaction">
   <xsd:complexType>
      <xsd:attribute name="tid" type="tns:non_empty_string" use="required"/>
      <xsd:attribute name="cpaID" type="xsd:IDREF" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="EndTransaction">
```

```
   <xsd:complexType>
      <xsd:attribute name="tid" type="tns:non_empty_string" use="required"/>
   </xsd:complexType>
</xsd:element>

<xsd:element name="DecisionPoint">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:Expression" minOccurs="1" maxOccurs="unbounded"/>
         <xsd:choice minOccurs="1" maxOccurs="unbounded">
            <xsd:element ref="tns:WhenTrue" />
            <xsd:element ref="tns:WhenFalse" />
         </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="description" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="id" type="xsd:ID" use="required" />
   </xsd:complexType>
</xsd:element>



<xsd:element name="SendMessage">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:Transport" />
         <xsd:element ref="tns:Packaging" minOccurs="0"/>
         <xsd:element ref="tns:SetMessage" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="description" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="toActor" type="xsd:IDREF" use="required" />
      <xsd:attribute name="id" type="xsd:ID" use="required" />
      <xsd:attribute name="cpaID" type="xsd:IDREF" use="optional" />
      <xsd:attribute name="transactionID" type="xsd:string" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="ReceiveMessage">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:Transport" />
         <xsd:element ref="tns:Packaging" minOccurs="0"/>
         <!-- the outputs will be read to the Variable in the specified order -->
         <xsd:choice maxOccurs="unbounded">
            <xsd:element ref="tns:VariableRef" />
            <xsd:element ref="tns:NullVariable" />
         </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="description" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="fromActor" type="xsd:IDREF" use="required" />
      <xsd:attribute name="cpaID" type="xsd:IDREF" use="optional" />
      <xsd:attribute name="id" type="xsd:ID" use="required" />
      <xsd:attribute name="transactionID" type="xsd:string" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="AskTestData">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:TestQuestion" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="toActor" type="xsd:IDREF" use="required" />
      <xsd:attribute name="description" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="id" type="xsd:ID" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="TestQuestion">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:VariableRef"/>
      </xsd:sequence>
      <xsd:attribute name="description" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="question" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="id" type="tns:non_empty_string" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="ListenMessage">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:Transport" />
         <xsd:element ref="tns:Packaging" minOccurs="0"/>
         <!-- the outputs will be read to the Variable in the specified order -->
         <xsd:choice maxOccurs="unbounded">
            <xsd:element ref="tns:VariableRef" />
            <xsd:element ref="tns:NullVariable" />
         </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="description" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="fromActor" type="xsd:IDREF" use="required" />
      <xsd:attribute name="toActor" type="xsd:IDREF" use="required" />
      <xsd:attribute name="cpaID" type="xsd:IDREF" use="optional" />
      <xsd:attribute name="id" type="xsd:ID" use="required" />
      <xsd:attribute name="transactionID" type="xsd:string" use="required" />
   </xsd:complexType>
```

```
</xsd:element>

<xsd:element name="SetMessage">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element minOccurs="0" maxOccurs="unbounded" ref="tns:Content" />
      </xsd:sequence>
      <xsd:attribute name="description" type="tns:non_empty_string" use="optional" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="TestAssertion">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:choice maxOccurs="unbounded">
            <xsd:element ref="tns:VerifyContent" />
            <xsd:element ref="tns:ValidateContent" />
         </xsd:choice>
         <xsd:element minOccurs="0" maxOccurs="unbounded" ref="tns:WhenTrue" />
         <xsd:element minOccurs="0" maxOccurs="unbounded" ref="tns:WhenFalse" />
      </xsd:sequence>
      <xsd:attribute name="description" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="id" type="xsd:ID" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="ValidateContent">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:Expression" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="validationAdapter" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="schemaLocation" type="xsd:anyURI" use="optional" />
      <xsd:attribute name="description" type="tns:non_empty_string" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="VerifyContent">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:Expression" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="verificationAdapter" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="verificationFunction" type="tns:non_empty_string" use="optional" />
      <xsd:attribute name="description" type="tns:non_empty_string" use="required" />
   </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="Assign">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:Expression"/>
      </xsd:sequence>
      <xsd:attribute name="append" type="tns:append_type" use="required"/>
      <xsd:attribute name="variableRef" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="id" type="xsd:ID" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="PreliminaryData">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:VariableRef"/>
      </xsd:sequence>
      <xsd:attribute name="description" type="tns:non_empty_string" use="required" />
      <xsd:attribute name="toActor" type="xsd:IDREF" use="optional" />
      <xsd:attribute name="type" type="tns:preliminary_data_type" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="Variable">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:Type" />
         <xsd:choice minOccurs="0">
            <xsd:element ref="tns:Value" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element ref="tns:InitiateValue"/>
         </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="description" type="tns:non_empty_string" use="optional" />
      <xsd:attribute name="name" type="xsd:ID" use="required" />
      <xsd:attribute name="mime_type" type="tns:non_empty_string" use="optional" />
      <xsd:attribute name="inMemory" type="xsd:boolean" use="optional" default="true"/>
   </xsd:complexType>
</xsd:element>

<xsd:element name="Packaging">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:ConfigurationAttribute" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attribute name="packagingHandler" type="tns:non_empty_string" use="required" />
   </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="Transport">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:ConfigurationAttribute" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attribute name="transportHandler" type="tns:non_empty_string" use="required" />
   </xsd:complexType>
</xsd:element>

<xsd:element name="ConfigurationAttribute">
   <xsd:complexType>
      <xsd:simpleContent>
         <xsd:extension base="tns:non_empty_string">
            <xsd:attribute name="name" type="tns:non_empty_string" use="required" />
            <xsd:attribute name="isRequired" type="xsd:boolean" use="optional" />
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
</xsd:element>


<xsd:element name="Content">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:choice>
            <xsd:element maxOccurs="unbounded" ref="tns:Reference" />
            <xsd:element maxOccurs="unbounded" ref="tns:VariableRef"/>
            <xsd:element maxOccurs="1" ref="tns:NullVariable" />
         </xsd:choice>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="Reference">
   <xsd:complexType>
      <xsd:attribute name="href" type="tns:non_empty_string" />
      <xsd:attribute name="isDynamicContent" type="xsd:boolean" use="optional" default="false"/>
      <xsd:attribute name="mime_type" type="tns:non_empty_string" use="optional"/>
   </xsd:complexType>
</xsd:element>

<xsd:element name="WhenTrue">
   <xsd:complexType>
      <xsd:choice>
         <xsd:element ref="tns:ThreadGroupRef" />
         <xsd:element ref="tns:Exit" />
      </xsd:choice>
   </xsd:complexType>
</xsd:element>

<xsd:element name="WhenFalse">
   <xsd:complexType>
      <xsd:choice>
         <xsd:element ref="tns:ThreadGroupRef" />
         <xsd:element ref="tns:Exit" />
      </xsd:choice>
   </xsd:complexType>
</xsd:element>

<xsd:element name="Exit">
   <xsd:complexType>
      <xsd:simpleContent>
         <xsd:extension base="tns:non_empty_string">
            <xsd:attribute name="description" type="tns:non_empty_string" use="required"/>
            <xsd:attribute name="onExit" type="tns:onExit_type" use="required"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
</xsd:element>

<xsd:element name="Expression">
   <xsd:complexType>
      <xsd:simpleContent>
         <xsd:extension base="tns:non_empty_string">
            <xsd:attribute name="namespaceBindings" type="tns:non_empty_string" use="optional"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
</xsd:element>

<xsd:element name="InitiateValue">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:ConfigurationAttribute" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attribute name="valueInitiator" type="tns:non_empty_string" use="required"/>
      <xsd:attribute name="valueID" type="tns:non_empty_string" use="required"/>
      <xsd:attribute name="description" type="tns:non_empty_string" use="optional"/>
   </xsd:complexType>
</xsd:element>

<xsd:element name="Title" type="tns:non_empty_string" />
<xsd:element name="Version" type="tns:non_empty_string" />
```

```
<xsd:element name="Maintainer" type="tns:non_empty_string" />
<xsd:element name="Location" type="xsd:anyURI" />
<xsd:element name="PublishDate" type="tns:non_empty_string" />
<xsd:element name="Status" type="tns:non_empty_string" />
<xsd:element name="StepDuration" type="xsd:integer" />
<xsd:element name="UploadDocument" type="tns:non_empty_string"/>
<xsd:element name="VariableRef" type="tns:non_empty_string" />
<xsd:element name="NullVariable" type="tns:NullVariableType"/>
<xsd:element name="Name" type="tns:non_empty_string" />
<xsd:element name="Type" type="tns:variable_type" />
<xsd:element name="FileURI" type="xsd:anyURI" />
<xsd:element name="Value" type="tns:non_empty_string" />
<xsd:element name="ActorUnderTest" type="xsd:ID" />
<xsd:element name="TestCaseCoordinator" type="xsd:IDREF" />
<xsd:element name="CPAType" type="tns:non_empty_string" />
<xsd:element name="CPAId" type="xsd:anyURI" />
<xsd:element name="TestCaseRefID" type="tns:non_empty_string" />
<xsd:element name="Description" type="tns:non_empty_string" />

<xsd:simpleType name="NullVariableType">
   <xsd:restriction base = "xsd:NMTOKEN">
      <xsd:enumeration value = "null"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name = "variable_type">
   <xsd:restriction base = "xsd:NMTOKEN">
      <xsd:minLength value="1" />
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="non_empty_string">
   <xsd:restriction base="xsd:string">
      <xsd:minLength value="1" />
   </xsd:restriction>
</xsd:simpleType>

<!-- no.append = simply overrides the old value -->
<!-- append.non.list = appends an atomic item   -->
<!--   to the end of the list                   -->
<!-- append.list = concats a list with the      -->
<!--   second...                                -->
<xsd:simpleType name = "append_type">
   <xsd:restriction base = "xsd:NMTOKEN">
      <xsd:enumeration value = "no_append"/>
      <xsd:enumeration value = "append_non_list"/>
      <xsd:enumeration value = "append_list"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name = "preliminary_data_type">
      <xsd:restriction base = "xsd:NMTOKEN">
         <xsd:enumeration value = "show"/>
         <xsd:enumeration value = "ask"/>
      </xsd:restriction>
   </xsd:simpleType>

   <!-- enumaration values should be extended        -->
   <xsd:simpleType name = "onExit_type">
      <xsd:restriction base = "xsd:NMTOKEN">
         <xsd:enumeration value = "exit_TestCase"/>
         <xsd:enumeration value = "exit_TestSuite"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```

A.2 TestBATN Validation Adapter Metadata XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   targetNamespace="adapters.model.testsuite.testframework.srdc.com"
   attributeFormDefault="unqualified"
   elementFormDefault="qualified"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:tss="model.testsuite.testframework.srdc.com"
   xmlns:tns="adapters.model.testsuite.testframework.srdc.com">

   <xsd:import namespace="model.testsuite.testframework.srdc.com"
      schemaLocation="TestSuiteSchema.xsd"/>

   <xsd:element name="Adapters">
      <xsd:complexType>
         <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="tns:ValidationAdapter"/>
            <xsd:element ref="tns:VerificationAdapter"/>
            <xsd:element ref="tns:ExternalAdapter"/>
         </xsd:choice>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="ValidationAdapter">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:AdapterAttribute" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="name" type="tss:non_empty_string" use="required"/>
         <xsd:attribute name="version" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="author" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="description" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="schemaRequired" type="xsd:boolean" use="required"/>
         <xsd:attribute name="class" type="tss:non_empty_string" use="required"/>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="VerificationAdapter">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:VerificationFunction" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="tns:AdapterAttribute" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="name" type="tss:non_empty_string" use="required"/>
         <xsd:attribute name="version" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="author" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="description" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="verificationFunctionRequired" type="xsd:boolean" use="required"/>
         <xsd:attribute name="class" type="tss:non_empty_string" use="required"/>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="ExternalAdapter">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:VerificationFunction" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="name" type="tss:non_empty_string" use="required"/>
         <xsd:attribute name="endpoint" type="xsd:anyURI" use="required"/>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="VerificationFunction">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:AdapterAttribute" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="name" type="tss:non_empty_string" use="required"/>
         <xsd:attribute name="description" type="tss:non_empty_string" use="optional"/>
      </xsd:complexType>
   </xsd:element>
```

```
   <xsd:element name="AdapterAttribute">
      <xsd:complexType>
         <xsd:attribute name="name" type="tss:non_empty_string" use="required"/>
         <xsd:attribute name="type" type="tss:variable_type" use="required"/>
         <xsd:attribute name="description" type="tss:non_empty_string" use="optional"/>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

A.3 TestBATN Messaging Adapter Metadata XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   targetNamespace="messagingHandlers.model.testsuite.testframework.srdc.com"
   attributeFormDefault="unqualified"
   elementFormDefault="qualified"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:tss="model.testsuite.testframework.srdc.com"
   xmlns:tns="messagingHandlers.model.testsuite.testframework.srdc.com">

   <xsd:import namespace="model.testsuite.testframework.srdc.com"
      schemaLocation="TestSuiteSchema.xsd"/>

   <xsd:element name="MessagingHandlers">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:MessagingHandler" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="MessagingHandler">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:ExpectedConfigurationAttributes" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="tns:InputArguments" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="tns:OutputArguments" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
         <xsd:attribute name="name" type="xsd:ID" use="required"/>
         <!-- used in find message, e.g HttpSender and HttpReceiver are complementary-->
         <!-- messaging handlers-->
         <xsd:attribute name="complementaryMessagingHandler" type="xsd:IDREF" use="optional"/>
         <xsd:attribute name="layer" type="tns:messagingLayer_type" use="required"/>
         <xsd:attribute name="role" type="tns:role" use="required"/>
         <xsd:attribute name="version" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="author" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="description" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="class" type="tss:non_empty_string" use="required"/>
      </xsd:complexType>
   </xsd:element>

   <!--         Provides a list of all required ConfigurationAttribute elements **>
   <**        Default values are also included in case the Test Writer fails to **>
   <**        provide them.                                                     **>
   <**        /null/ values (e.g. for http-code) indicate an obligation,        **>
   <**        i.e. the Test Writer is enforced to provide some value for that   **>
   <**        ConfigurationAttribute                                            -->
   <xsd:element name="ExpectedConfigurationAttributes">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tss:ConfigurationAttribute" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>

   <!--         The order is important!-->
   <xsd:element name="InputArguments">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:Argument" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>

   <!--         The order is important!-->
   <xsd:element name="OutputArguments">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:Argument" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>

   <!--         The name of the Argument may be chosen from a list   **>
   <**        We may consider making representationType optional:    **>
   <**        i.e. if not defined a corresponding object with [=name]-->
   <xsd:element name="Argument">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:ApplicableType" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="name" type="tss:non_empty_string" use="required"/>
         <xsd:attribute name="description" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="required" type="xsd:boolean" use="required"/>
       </xsd:complexType>
     </xsd:element>
     <xsd:element name="ApplicableType" type="xsd:string"/>
     <xsd:simpleType name = "messagingLayer_type">
       <xsd:restriction base = "xsd:NMTOKEN">
         <xsd:enumeration value = "TransportLayer"/>
         <xsd:enumeration value = "PackagingLayer"/>
         <xsd:enumeration value = "ContentLayer"/>
       </xsd:restriction>
     </xsd:simpleType>
     <xsd:simpleType name = "role">
       <xsd:restriction base = "xsd:NMTOKEN">
         <xsd:enumeration value = "Sender"/>
         <xsd:enumeration value = "Receiver"/>
       </xsd:restriction>
     </xsd:simpleType>
   </xsd:schema>
```

A.4 TestBATN Data Types XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  targetNamespace="datatypes.model.testsuite.testframework.srdc.com"
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tss="model.testsuite.testframework.srdc.com"
  xmlns:tns="datatypes.model.testsuite.testframework.srdc.com">

  <xsd:import namespace="model.testsuite.testframework.srdc.com"
    schemaLocation="TestSuiteSchema.xsd"/>

  <xsd:element name="TestFrameworkDataRegistration">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:RegisteredType" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="RegisteredType">
    <xsd:complexType>
      <xsd:sequence>
```

```
            <xsd:element ref="tns:InitFromVariable" />
            <xsd:element ref="tns:InitFromMessage" />
         </xsd:sequence>
         <xsd:attribute name="name" type="tss:non_empty_string" use="required" />
         <xsd:attribute name="class" type="tss:non_empty_string" use="required" />
         <xsd:attribute name="serializable" type="xsd:boolean" use="required" />
         <xsd:attribute name="minSize" type="tns:rangeType" use="required" />
         <xsd:attribute name="maxSize" type="tns:rangeType" use="required" />
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="InitFromVariable">
      <xsd:complexType>
         <xsd:attribute name="nonURIAllowed" type="xsd:boolean" use="required" />
         <xsd:attribute name="URIAllowed" type="xsd:boolean" use="required" />
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="InitFromMessage">
      <xsd:complexType>
         <xsd:attribute name="nonURIAllowed" type="xsd:boolean" use="required" />
         <xsd:attribute name="URIAllowed" type="xsd:boolean" use="required" />
      </xsd:complexType>
   </xsd:element>

   <xsd:simpleType name="rangeType">
      <xsd:union>
         <xsd:simpleType>
            <xsd:restriction base="xsd:integer"/>
         </xsd:simpleType>
         <xsd:simpleType>
            <xsd:restriction base = "xsd:string">
               <xsd:enumeration value = "unbounded"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:union>
   </xsd:simpleType>
</xsd:schema>
```

A.5 TestBATN Value Initiators Metadata XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   targetNamespace="valueInitiator.model.testsuite.testframework.srdc.com"
   attributeFormDefault="unqualified"
   elementFormDefault="qualified"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:tss="model.testsuite.testframework.srdc.com"
   xmlns:mh="messagingHandlers.model.testsuite.testframework.srdc.com"
   xmlns:adp="adapters.model.testsuite.testframework.srdc.com"
   xmlns:tns="valueInitiator.model.testsuite.testframework.srdc.com">

   <xsd:import namespace="model.testsuite.testframework.srdc.com"
      schemaLocation="TestSuiteSchema.xsd"/>
   <xsd:import namespace="messagingHandlers.model.testsuite.testframework.srdc.com"
      schemaLocation="MessagingHandlers.xsd"/>
   <xsd:import namespace="adapters.model.testsuite.testframework.srdc.com"
      schemaLocation="Adapters.xsd"/>

   <xsd:element name="ValueInitiators">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="tns:ValueInitiator" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="ValueInitiator">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="mh:ExpectedConfigurationAttributes" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="adp:AdapterAttribute" minOccurs="1" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="name" type="xsd:ID" use="required"/>
         <xsd:attribute name="version" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="author" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="description" type="tss:non_empty_string" use="optional"/>
         <xsd:attribute name="class" type="tss:non_empty_string" use="required"/>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

APPENDIX B

SAMPLE METADATA FOR TESTBATN ADAPTORS

B.1 Sample Metadata for TestBATN Validation Adapters

```
<?xml version="1.0" encoding="UTF-8"?>
<Adapters
   xmlns="adapters.model.testsuite.testframework.srdc.com">
   <!-- XML Schema Validation Adapter -->
   <ValidationAdapter
      name="xsdadapter"
      version="0.9"
      author="SRDC-Team"
      description="Validates the XML document against an XSD schema"
      schemaRequired="true"
      class="com.srdc.testframework.evaladapter.XSDValidator">
      <AdapterAttribute name="Content" type="xml" description="The XML content to be syntactically evaluated against the schema"/>
   </ValidationAdapter>
   <!-- HL7 v2 Validation adapter (working on XML serialization of V2 messages)-->
   <ValidationAdapter
      name="hl7xmladapter"
      version="1.1"
      author="SRDC-Team"
      description="Validates the XML encoded HL7 message"
      schemaRequired="true"
      class="com.srdc.testframework.evaladapter.HL7XmlMessageValidator">
      <AdapterAttribute name="Content" type="xml" description="The XML content to be syntactically evaluated against the hl7 profile"/>
   </ValidationAdapter>
   <!-- Regular Expression Validation Adapter -->
   <ValidationAdapter
      name="regexpradapter"
      version="0.9"
      author="SRDC-Team"
      description="Performs validation against a regular expression (defined through the schema)"
      schemaRequired="true"
      class="com.srdc.testframework.evaladapter.RegularExprValidator">
      <AdapterAttribute name="Content" type="string" description="content to be validated against the schema (regexp)"/>
   </ValidationAdapter>
   <!-- Schematron Validation Adapter-->
   <VerificationAdapter
      name="schematronadapter"
      version="0.9"
      author="SRDC-Team"
      verificationFunctionRequired="false"
      class="com.srdc.testframework.evaladapter.SchematronValidator">
      <AdapterAttribute name="resource" type="xml" description="resource to be verified against the schematron"/>
      <AdapterAttribute name="verificationDocumentLoacation" type="string" description="location of the schematron"/>
```

```
  </VerificationAdapter>
  <!-- XPATH Validation Adapter-->
  <VerificationAdapter
    name="xpathadapter"
    version="0.9"
    author="SRDC-Team"
    verificationFunctionRequired="false"
    class="com.srdc.testframework.evaladapter.XPathValidator">
    <AdapterAttribute name="resource" type="xml" description="resource to be verified using xpath"/>
    <AdapterAttribute name="xpathExpr" type="expr" description="the XPATH expression itself"/>
  </VerificationAdapter>
  <!-- A sample proprietary validation adapter providing several functions for validation-->
  <VerificationAdapter
    name="expressionadapter"
    version="0.9"
    author="SRDC-Team"
    verificationFunctionRequired="true"
    class="com.srdc.testframework.evaladapter.BasicExpressionValidator">
    <VerificationFunction name="Member" description="?(element [member-of] group)">
      <AdapterAttribute name="value" type="string" description="optional"/>
      <AdapterAttribute name="group" type="list_string" description="optional"/>
    </VerificationFunction>

    <VerificationFunction name="Equal" description="?(element1 == element2)">
      <AdapterAttribute name="value1" type="string"/>
      <AdapterAttribute name="value2" type="string"/>
    </VerificationFunction>

    <VerificationFunction name="ListEqual" description="?contains(list1, list2) and ?equals(list1, list2)">
      <AdapterAttribute name="value1" type="list_string"/>
      <AdapterAttribute name="value2" type="list_string"/>
    </VerificationFunction>
    <VerificationFunction name="Contains" description="?contains(element1, element2)">
      <AdapterAttribute name="value1" type="string"/>
      <AdapterAttribute name="value2" type="string"/>
    </VerificationFunction>
  </VerificationAdapter>
  <!-- Validation adapter for Turkish National Health System's Code Systems -->
  <VerificationAdapter
    name="SKRSALLValidator"
    version="0.9"
    author="SRDC-Team"
    verificationFunctionRequired="false"
    class="com.srdc.testframework.evaladapter.SKRSALLValidator">
    <AdapterAttribute name="hl7CodeListElement" type="list_xml" description="'hl7:code' elemani listesi, XPATH ile tum bu elemanlar
list_xml haline getirilebilir"/>
  </VerificationAdapter>
  <!-- External Validation Adapters running as a web service -->
  <ExternalAdapter name="sqlverifier" endpoint="http://144.122.230.23/SQLVerificationService"/>
  <ExternalAdapter name="syslogverifier" endpoint="http://144.122.230.23/SyslogVerificationService"/>
</Adapters>
```

B.2 Sample Metadata for TestBATN Messaging Adapters

```
<?xml version="1.0" encoding="UTF-8"?>
<MessagingHandlers
  xmlns="messagingHandlers.model.testsuite.testframework.srdc.com"
  xmlns:tss="model.testsuite.testframework.srdc.com">
  <MessagingHandler
```

```
   name="TCPReceiver"
   complementaryMessagingHandler="TCPSender"
   layer="TransportLayer"
   role="Receiver"
   version="0.9"
   author="SRDC-Team"
   class="com.srdc.testframework.testengine.transport.receivers.TCPReceiver">
   <ExpectedConfigurationAttributes>
      <tss:ConfigurationAttribute name="stopCharacter">null</tss:ConfigurationAttribute>
   </ExpectedConfigurationAttributes>
   <InputArguments/>
   <OutputArguments>
      <Argument required="true" name="TCP stream" description="stream received from the TCP connection.">
         <ApplicableType>string</ApplicableType>
         <ApplicableType>xml</ApplicableType>
         <ApplicableType>bytes</ApplicableType>
      </Argument>
   </OutputArguments>
</MessagingHandler>
<MessagingHandler
   name="TCPSender"
   complementaryMessagingHandler="TCPReceiver"
   layer="TransportLayer"
   role="Sender"
   version="0.9"
   author="SRDC-Team"
   class="com.srdc.testframework.testengine.transport.senders.TCPSender">
   <ExpectedConfigurationAttributes/>
   <InputArguments>
      <Argument required="true" name="TCP stream" description="stream to be sent via the TCP connection.">
         <ApplicableType>string</ApplicableType>
         <ApplicableType>xml</ApplicableType>
         <ApplicableType>bytes</ApplicableType>
      </Argument>
   </InputArguments>
   <OutputArguments/>
</MessagingHandler>
<MessagingHandler
   name="HTTPReceiver"
   complementaryMessagingHandler="HTTPSender"
   layer="TransportLayer"
   role="Receiver"
   version="0.9"
   author="SRDC-Team"
   class="com.srdc.testframework.testengine.transport.receivers.HTTPReceiver">
   <ExpectedConfigurationAttributes>
      <tss:ConfigurationAttribute name="method">null</tss:ConfigurationAttribute>
   </ExpectedConfigurationAttributes>
   <InputArguments/>
   <OutputArguments>
      <Argument required="true" name="http-header" description="The HTTP-HEADER extracted from the transaction.">
         <ApplicableType>string</ApplicableType>
      </Argument>
      <Argument required="true" name="http-body" description="The HTTP-BODY extracted from the transaction.">
         <ApplicableType>string</ApplicableType>
         <ApplicableType>xml</ApplicableType>
         <ApplicableType>bytes</ApplicableType>
      </Argument>
   </OutputArguments>
</MessagingHandler>
<MessagingHandler
   name="HTTPSender"
   complementaryMessagingHandler="HTTPReceiver"
   layer="TransportLayer"
   role="Sender"
   version="0.9"
   author="SRDC-Team"
   class="com.srdc.testframework.testengine.transport.senders.HTTPSender">

   <ExpectedConfigurationAttributes>
      <tss:ConfigurationAttribute name="method">null</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="Content-Type">null</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="code">null</tss:ConfigurationAttribute>
   </ExpectedConfigurationAttributes>

   <InputArguments>
      <Argument required="false" name="http-header" description="The HTTP-Header used in the transaction.">
         <ApplicableType>string</ApplicableType>
      </Argument>
      <Argument required="true" name="http-body" description="The HTTP-Body used in the transaction.">
         <ApplicableType>string</ApplicableType>
      </Argument>

   </InputArguments>
   <OutputArguments/>

</MessagingHandler>
<MessagingHandler
   name="SyslogReceiver"
   complementaryMessagingHandler="SyslogSender"
   layer="PackagingLayer"
   role="Receiver"
   version="0.9"
   author="SRDC-Team"
   class="com.srdc.testframework.testengine.transport.receivers.SyslogRFC3164Receiver">
   <ExpectedConfigurationAttributes/>
   <InputArguments/>
   <OutputArguments>
      <Argument required="true" name="syslog header" description="Syslog Header (RFC316)">
         <ApplicableType>string</ApplicableType>
      </Argument>
      <Argument required="true" name="syslog body" description="Syslog Body (any type)">
         <ApplicableType>string</ApplicableType>
         <ApplicableType>xml</ApplicableType>
      </Argument>
   </OutputArguments>
</MessagingHandler>
<MessagingHandler
   name="SyslogSender"
   complementaryMessagingHandler="SyslogReceiver"
   layer="PackagingLayer"
   role="Sender"
   version="0.9"
   author="SRDC-Team"
   class="com.srdc.testframework.testengine.transport.senders.SyslogRFC3164Sender">
   <ExpectedConfigurationAttributes/>
   <InputArguments/>
   <OutputArguments>
      <Argument required="true" name="syslog body" description="Syslog Body (any type)">
         <ApplicableType>string</ApplicableType>
         <ApplicableType>xml</ApplicableType>
      </Argument>
   </OutputArguments>
</MessagingHandler>
<MessagingHandler
   name="SOAPReceiver"
```

```
      layer="PackagingLayer"
      role="Receiver"
      version="0.9"
      author="SRDC-Team"
      class="com.srdc.testframework.testengine.transport.receivers.SOAPReceiver">
      <ExpectedConfigurationAttributes/>
      <InputArguments/>
      <OutputArguments>
         <Argument required="true" name="SOAP-Header content" description="text/xml">
            <ApplicableType>string</ApplicableType>
            <ApplicableType>xml</ApplicableType>
         </Argument>
         <Argument required="true" name="SOAP-Body content" description="text/xml">
            <ApplicableType>string</ApplicableType>
            <ApplicableType>xml</ApplicableType>
         </Argument>
      </OutputArguments>
   </MessagingHandler>
   <MessagingHandler
      name="SOAPSender"
      layer="PackagingLayer"
      role="Sender"
      version="0.9"
      author="SRDC-Team"
      class="com.srdc.testframework.testengine.transport.senders.SOAPSender">
      <ExpectedConfigurationAttributes/>
      <InputArguments>
         <Argument required="true" name="SOAP-Body content" description="text/xml">
            <ApplicableType>string</ApplicableType>
            <ApplicableType>xml</ApplicableType>
         </Argument>
      </InputArguments>
      <OutputArguments/>
   </MessagingHandler>
</MessagingHandlers>
```

B.3 Sample Metadata for TestBATN Data Types

```
<?xml version="1.0" encoding="UTF-8"?>
<TestFrameworkDataRegistration xmlns="datatypes.model.testsuite.testframework.srdc.com">
   <RegisteredType name="string" class="com.srdc.testframework.testengine.data.holder.StringImpl"
serializable="true" minSize="1" maxSize="1">
      <InitFromVariable nonURIAllowed="true" URIAllowed="false" />
      <InitFromMessage nonURIAllowed="true" URIAllowed="false" />
   </RegisteredType>
   <RegisteredType name="xml" class="com.srdc.testframework.testengine.data.holder.XMLImpl"
serializable="true" minSize="1" maxSize="1">
      <InitFromVariable nonURIAllowed="false" URIAllowed="true" />
      <InitFromMessage nonURIAllowed="true" URIAllowed="false" />
   </RegisteredType>
   <RegisteredType name="bytes" class="com.srdc.testframework.testengine.data.holder.RawDataImpl"
serializable="true" minSize="1" maxSize="1">
      <InitFromVariable nonURIAllowed="false" URIAllowed="true" />
      <InitFromMessage nonURIAllowed="true" URIAllowed="false" />
   </RegisteredType>
   <RegisteredType name="expr" class="com.srdc.testframework.testengine.data.holder.ExprImpl"
serializable="true" minSize="1" maxSize="1">
      <InitFromVariable nonURIAllowed="false" URIAllowed="false" />
      <InitFromMessage nonURIAllowed="false" URIAllowed="false" />
   </RegisteredType>
   <RegisteredType name="object" class="com.srdc.testframework.testengine.data.holder.ObjectImpl"
serializable="true" minSize="1" maxSize="1">
      <InitFromVariable nonURIAllowed="false" URIAllowed="true" />
      <InitFromMessage nonURIAllowed="true" URIAllowed="false" />
   </RegisteredType>
   <RegisteredType name="null" class="com.srdc.testframework.testengine.data.holder.EmptyTestFrameworkDataImpl"
serializable="true" minSize="1" maxSize="1">
      <InitFromVariable nonURIAllowed="false" URIAllowed="false" />
      <InitFromMessage nonURIAllowed="false" URIAllowed="false" />
   </RegisteredType>
   <RegisteredType name="dicom" class="com.srdc.testframework.dicom.datatype.DicomObjectImpl"
serializable="true" minSize="1" maxSize="1">
      <InitFromVariable nonURIAllowed="false" URIAllowed="true" />
      <InitFromMessage nonURIAllowed="true" URIAllowed="false" />
   </RegisteredType>
</TestFrameworkDataRegistration>
```

APPENDIX C

EXAMPLE TEST SCENARIO DEFINITION FROM TURKISH NHIS TEST SUITES

C.1 Test Suite Definition for NHIS Muayene Web Services

```
<TestSuite xmlns="model.testsuite.testframework.srdc.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   id="Muayene_TestSuite">
   <MetaData>
      <Title>Saglik Bakanligi - Muayene Veri Akisi Testleri</Title>
      <Version>0.1</Version>
      <Maintainer>SRDC Team</Maintainer>
      <Location>Ankara,Turkey</Location>
      <PublishDate>24/03/2008</PublishDate>
      <Status>DRAFT</Status>
      <Description>Muayene MSVS Bildirimi Uyumluluk Testleri</Description>
   </MetaData>
   <Variables>
   </Variables>
   <TestCaseRefID>Muayene_HastaCikisMSVSAnlamsalTestSenaryosu</TestCaseRefID>
   <TestCaseRefID>Muayene_HastaKabulMSVSAnlamsalTestSenaryosu</TestCaseRefID>
   <TestCaseRefID>Muayene_MuayeneMSVSAnlamsalTestSenaryosu</TestCaseRefID>
   <TestCaseRefID>Muayene_ReceteMSVSAnlamsalTestSenaryosu</TestCaseRefID>
   <TestCaseRefID>Muayene_TetkikSonucuMSVSAnlamsalTestSenaryosu</TestCaseRefID>
   <TestCaseRefID>Muayene_SorgulamaServisiTestSenaryosu</TestCaseRefID>
   <TestCaseRefID>Muayene_TemelBirlikteIslerlikTestSenaryosu</TestCaseRefID>
   <TestCaseRefID>Muayene_TemelTestSenaryosu</TestCaseRefID>
   <TestCaseRefID>Muayene_TemelTestSenaryosu_Recete</TestCaseRefID>
   <TestCaseRefID>Muayene_TemelTestSenaryosu_TetkikSonuc</TestCaseRefID>
   <TestCaseRefID>Muayene_IptalServisiTestSenaryosu</TestCaseRefID>
   <TestCaseRefID>Muayene_GuncellemeServisiTestSenaryosu</TestCaseRefID>
</TestSuite>
```

C.2 Test Case Definition for the Basic Test Scenario for NHIS Muayene Web Service

```
<?xml version="1.0" encoding="UTF-8"?>
<TestCase xmlns="model.testsuite.testframework.srdc.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   id = "Muayene_TemelTestSenaryosu" description = "Muayene Bildirimi baz senaryo testi.">
   <MetaData>
      <Title>USBS - Muayene Bildirimi baz senaryo testi</Title>
      <Version>0.1</Version>
      <Maintainer>SRDC Ltd.</Maintainer>
      <Location>Ankara,Turkiye</Location>
      <PublishDate>24/03/2008</PublishDate>
      <Status>TASLAK</Status>
      <Description>Bu test senaryosu Muayene Bildiriminin temel gereksinimlerini test eder. Bu temel gereksinimler;
         - bildirimin sozdizimsel yapisinin dogru olmasi,
         - gonderilen SOAP mesajinin gecerli olmasi,
         - belirlenmis WS-UsernameToken gereksinimlerine uygunluk,
         - kullanici adi ve sifrenin gecerli olmasi,
         - Muayene Bildiriminde gecen kod referanslarinin gecerli olmasi,
         - ve bildirimin belirlenen is kurallarina uygun olmasidir.
         Bu test senaryosunu herhangi bir Muayene Bildiriminizi test etmek icin kullanabilirsiniz.
      </Description>
   </MetaData>
   <!-- Parties Participating the Test-->
   <ActorUnderTest>USBSSimulator</ActorUnderTest>
   <ActorUnderTest>USBSIstemci</ActorUnderTest>
   <!--USBSSimulator is our TestEngine-->
   <TestCaseCoordinator>USBSSimulator</TestCaseCoordinator>
   <!--Configurations-->
   <ConfigurationGroup>
      <!-- Configuration between SagilikNETServer and USBSClient -->
      <CPA id="Etkilesim1" fromActor="USBSIstemci" toActor="USBSSimulator">
         <CPAType>USBS Muayene Veri Akisi</CPAType>
         <CPAId>http://www.srdc.metu.edu.tr/moh/cpa/Muayene.xml</CPAId>
      </CPA>
      <PreliminaryData toActor="USBSIstemci" description="Sisteminizin bu test icin kullanacagi kullanici adi bilgisini giriniz." type="ask">
         <VariableRef>KullaniciAdi</VariableRef>
      </PreliminaryData>
      <PreliminaryData toActor="USBSIstemci" description="Sisteminizin bu test icin kullanacagi sifre bilgisini giriniz." type="ask">
         <VariableRef>Sifre</VariableRef>
      </PreliminaryData>
   </ConfigurationGroup>
   <Variables>
      <Variable name="httpHeader" description="HTTP Header parcalarini saklamak icin kullanilacak degisken">
         <Type>string</Type>
      </Variable>
      <Variable name="soapHeader" description="SOAP Header parcalarini saklamak icin kullanilacak degisken">
         <Type>xml</Type>
      </Variable>
      <Variable name="soapBody" description="SOAP Body parcalarini saklamak icin kullanilacak degisken">
         <Type>xml</Type>
      </Variable>
      <Variable name="MSVSbildirimi" description="USBS HL7 Muayene Bildirim Mesaji">
         <Type>xml</Type>
      </Variable>
      <Variable name="hl7CodeElementList" description="USBS HL7 Muayene Mesaji icindeki hl7:code elemanlarinin listesi">
         <Type>list_xml</Type>
      </Variable>
      <Variable name="KullaniciAdi" description="USBS SOAP mesajlarinda authentication icin kullanilan WS Username-token profilindeki kullanici
adi(username)">
         <Type>string</Type>
      </Variable>
      <Variable name="Sifre" description="USBS SOAP mesajlarinda authentication icin kullanilan WS Username-token profilindeki sifre(password)">
         <Type>string</Type>
      </Variable>
      <!-- Some details for MSVS Message -->
     <Variable name="messageID" description="USBS Bildirim mesajlarinda ilinti(correlation) icin kullanilan mesaj ID">
         <Type>string</Type>
         <Value>Unknown</Value>
      </Variable>
      <Variable name="senderID" description="USBS Bildirim mesajlarinda gondereni tanimlamak icin kullanilan Gonderen ID">
```

```
         <Type>string</Type>
         <Value>Unknown</Value>
      </Variable>
      <Variable name="messageTime" description="Simulatorun gonderecegi mesajlarda creationTime elemani icin kullanilir">
         <Type>string</Type>
      </Variable>
      <Variable name="ackDetailCode" description="Simulatorun gonderecegi tasdik(acknowledgement) mesajlarindaki hata(error) kodu">
         <Type>string</Type>
      </Variable>
      <Variable name="ackDetailDisplayName" description="Simulatorun gonderecegi tasdik(acknowledgement) mesajlarindaki hata(error) kodu ">
         <Type>string</Type>
      </Variable>
      <Variable name="messageUUID" description="Simulatorun gonderecegi tasdik(acknowledgement) mesajlarindaki UUID kodu ">
         <Type>string</Type>
      </Variable>
      <Variable name="ackTypeCode" description="CA ya da CE">
         <Type>string</Type>
         <Value>CA</Value>
      </Variable>
    </Variables>
        <ThreadGroup name="TG01" description="Main ThreadGroup">
           <Thread name = "TH01" description="Muayene Bildirimi Baz Senaryosunun temel adimlari.">
              <BeginTransaction tid="T01" cpaID="Etkilesim1"/>
              <ReceiveMessage fromActor="USBSIstemci" id="R01" description = "Bu Test Senaryosu surecinde olusan Muayene Bildirimini,
USBSSimulator'unun konfigurasyon kisminda belirtilen ag adresine gonderiniz."  transactionID="T01">
                 <Transport transportHandler="HTTPReceiver"/>
                 <Packaging packagingHandler="SOAPReceiver"/>
                 <VariableRef>soapHeader</VariableRef>
                 <VariableRef>soapBody</VariableRef>
              </ReceiveMessage>
              <!-- By convention, store the message content into the our main message element -->
              <Assign id="ASGN_01" variableRef="MSVSbildirimi" append="no_append">
                 <Expression>$soapBody</Expression>
              </Assign>
              <!-- WSUserName/Password Compliance Checking-->
              <ThreadGroupRef nameRef="TG02"/>

              <!-- XML Schema Validation -->
              <TestAssertion id="TA2" description = "Gonderilen Muayene Bildirimi sozdizimsel yapisinin belirlenen Muayene XML Semasina gore
test edilmesi">
                 <ValidateContent validationAdapter="xsdadapter" schemaLocation="testsuites/MOH_Muayene/MoHServer/XSDSchemas/muayene/
MCCI_IN000001TR01.xsd" description="Gonderilen Muayene Bildirimi sozdizimsel yapisinin belirlenen Muayene XML Semasina gore test edilmesi">
                    <Expression>$MSVSbildirimi</Expression>
                 </ValidateContent>
                 <VerifyContent verificationAdapter="xpathadapter" description="Gonderilen bildirimdeki Muayene Verisetinin icinde bir adet
ANATANI veri elemani bulunmasi gerekir.">
                    <Expression>$MSVSbildirimi</Expression>
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">count(//hl7:examinationDataset//hl7:diagnosisSection) =
count(//hl7:examinationDataset//hl7:diagnosisSection//hl7:diagnosis/hl7:code[@codeSystem='2.16.840.1.113883.3.129.2.2.6'][@code='ANATANI'])
</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="xpathadapter" description="Gonderilen bildirimdeki her bir Recete Verisetinin icinde
bir adet ANATANI veri elemani bulunmasi gerekir.">
                    <Expression>$MSVSbildirimi</Expression>
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">count(//hl7:prescriptionDataset//hl7:presDiagnosisSection) =
count(//hl7:prescriptionDataset//hl7:presDiagnosisSection//hl7:presDiagnosis/hl7:code[@codeSystem='2.16.840.1.113883.3.129.2.2.6'][@code='ANATANI'])
</Expression>
                 </VerifyContent>
                 <!-- If syntactically wrong, send the error message defined in Integration Document and exit from Test-->
                 <WhenFalse>
                    <ThreadGroupRef nameRef = "TG05"/>
                 </WhenFalse>
              </TestAssertion>
              <!-- Tests on reference codes that MSVS message includes -->
              <!-- get the list of hl7:code elements in the MSVS message-->
              <Assign id="ASGN_02" variableRef="hl7CodeElementList" append="no_append">
                 <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">$MSVSbildirimi//*[@code][@codeSystem]</Expression>
              </Assign>
              <TestAssertion id="TA3" description = "Gonderilen Muayene Bildiriminde gecen tum HL7 code referanslarinin SaglikNET SKRS sistemine
uygunlugunun test edilmesi.">
                 <VerifyContent verificationAdapter="SKRSALLValidator" description="Gonderilen Muayene Bildiriminde gecen tum HL7 code
referanslarinin USBS SKRS sistemine uygunlugunun test edilmesi.">
                    <Expression>$hl7CodeElementList</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="SKRSValidator" description="Bildirimde gecen 'Kurum' kodunun SKRS sistemine uygunlugunun
test edilmesi.">
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">'2.16.840.1.113883.3.129.1.1.6'</Expression>
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">$MSVSbildirimi//hl7:representedHealthcareOrganization/hl7:id/
@extension</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="SKRSValidator" description="Bildirimde Muayene Verisetinde bulunabilecek 'Tetkik Istenen
Kurum' kodlarinin SKRS sistemine uygunlugunun test edilmesi.">
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">'2.16.840.1.113883.3.129.1.1.6'</Expression>
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">$MSVSbildirimi//hl7:examinationDataset//hl7:testSection//hl7:test
/hl7:performer/hl7:performerOrganization/hl7:id/@extension</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="SKRSValidator" description="Bildirimde Yenidogan Kayit Verisetinde bulunabilecek 'Dogum
Sirasi' kodunun SKRS sistemine uygunlugunun test edilmesi.">
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">'32727faa-185d-4ecb-a837-b70cf8e300f8'</Expression>
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">$MSVSbildirimi//hl7:babyPatientRole/hl7:patientBaby/hl7:multiple
BirthOrderNumber/@value</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="SKRSValidator" description="Bildirimde Recete Verisetinde bulunabilecek 'Ilac Kullanim
Periyodu Birimi' kodunun SKRS sistemine uygunlugunun test edilmesi.">
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">'4169b0db-6445-4222-9c0d-9bb71343b420'</Expression>
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">$MSVSbildirimi//hl7:prescriptionDataset//hl7:medicationSection
//hl7:substanceAdministration/hl7:effectiveTime/hl7:period/@unit</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="SKRSValidator" description="Bildirimde Tetkik Sonucu Verisetinde bulunabilecek 'Tetkik
Yapan Kurum' kodunun SKRS sistemine uygunlugunun test edilmesi.">
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">'2.16.840.1.113883.3.129.1.1.6'</Expression>
                    <Expression namespaceBindings = "{hl7 = urn:hl7-org:v3}">$MSVSbildirimi//hl7:testResultDataset//hl7:testResultSection
//hl7:testResultOrganizer/hl7:performer/hl7:studyOrganization/hl7:id/@extension</Expression>
                 </VerifyContent>
              </TestAssertion>

              <!-- Muayene MSVS Testing of Business Rules -->
              <TestAssertion id="TA4" description = "Gonderilen Muayene Bildiriminin, bu bildirim icin belirlenen is kurallari goz onune alinarak
 test edilmesi">
                 <VerifyContent verificationAdapter="schematronadapter" description="Gonderilen Muayene Bildiriminin, bu bildirim icin belirlenen
 is kurallari goz onune alinarak test edilmesi">
                    <Expression>$MSVSbildirimi</Expression>
                    <Expression>'testsuites/MOH_Muayene/MoHServer/Schematrons/MuayeneSchematronNs.xml'</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="schematronadapter" description="Gonderilen Muayene Bildiriminin Muayene kisminin, bu bildirim
 icin belirlenen is kurallari goz onune alinarak test edilmesi">
                    <Expression>$MSVSbildirimi</Expression>
                    <Expression>'testsuites/MOH_Muayene/MoHServer/Schematrons/MuayeneMSVS.xml'</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="schematronadapter" description="Gonderilen Muayene Bildiriminin Hasta Kabul kisminin, bu
bildirim icin belirlenen is kurallari goz onune alinarak test edilmesi">
                    <Expression>$MSVSbildirimi</Expression>
                    <Expression>'testsuites/MOH_Muayene/MoHServer/Schematrons/HastaKabulMSVS.xml'</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="schematronadapter" description="Gonderilen Muayene Bildiriminin Hasta Cikis kisminin, bu
```

```
bildirim icin belirlenen is kurallari goz onune alinarak test edilmesi">
                    <Expression>$MSVSbildirimi</Expression>
                    <Expression>'testsuites/MOH_Muayene/MoHServer/Schematrons/HastaCikisMSVS.xml'</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="schematronadapter" description="Gonderilen Muayene Bildiriminin varsa Recete kisminin, bu
bildirim icin belirlenen is kurallari goz onune alinarak test edilmesi">
                    <Expression>$MSVSbildirimi</Expression>
                    <Expression>'testsuites/MOH_Muayene/MoHServer/Schematrons/ReceteMSVS.xml'</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="schematronadapter" description="Gonderilen Muayene Bildiriminin varsa Tetkik Sonuc kisminin,
bu bildirim icin belirlenen is kurallari goz onune alinarak test edilmesi">
                    <Expression>$MSVSbildirimi</Expression>
                    <Expression>'testsuites/MOH_Muayene/MoHServer/Schematrons/TetkikSonucMSVS.xml'</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="schematronadapter" description="Gonderilen Muayene Bildiriminin varsa Vatandas Yabanci Kayit
kisminin, bu bildirim icin belirlenen is kurallari goz onune alinarak test edilmesi">
                    <Expression>$MSVSbildirimi</Expression>
                    <Expression>'testsuites/MOH_Muayene/MoHServer/Schematrons/VatandasYabanciKayitMSVS.xml'</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="schematronadapter" description="Gonderilen Muayene Bildiriminin varsa Yenidogan Kayit kisminin,
 bu bildirim icin belirlenen is kurallari goz onune alinarak test edilmesi">
                    <Expression>$MSVSbildirimi</Expression>
                    <Expression>'testsuites/MOH_Muayene/MoHServer/Schematrons/YenidoganKayitMSVS.xml'</Expression>
                 </VerifyContent>
              </TestAssertion>
              <!--Send the acknowledgement with CA typeCode-->
              <ThreadGroupRef nameRef = "TG06"/>
           </Thread>
        </ThreadGroup>

        <ThreadGroup name="TG02" description="WS-Username/Token Profiline uygunlugun test edilmesi">
           <Thread name = "TH02" description="WS-Username/Token Profiline uygunlugun test edilmesi">
              <TestAssertion id="TA5" description = "Gonderilen Muayene Bildirimi SOAP mesajinin WS-Username-Token Profiline uygunlugunun test
edilmesi">
                 <VerifyContent verificationAdapter="xpathadapter" description="SOAP Header kisminda 'wsse:Security' elemaninin bulunmasi
gerekir.">
                    <Expression>$soapHeader</Expression>
                    <Expression namespaceBindings = "{soap = http://schemas.xmlsoap.org/soap/envelope/, wsse = http://docs.oasis-open.org/wss/
2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd, wsu = http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd}">
count(/soap:Header/wsse:Security) = 1</Expression>
                 </VerifyContent>

                 <VerifyContent verificationAdapter="xpathadapter" description="'wsse:Security' elemaninin 'mustUnderstand' ozniteligi(attribute)
 belirtilmemeli veya belirtilmisse degeri 0 olmalidir.">
                    <Expression>$soapHeader</Expression>
                    <Expression namespaceBindings = "{soap = http://schemas.xmlsoap.org/soap/envelope/, wsse = http://docs.oasis-open.org/wss/
2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd, wsu = http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd}">
count(/soap:Header/wsse:Security[@soap:mustUnderstand]) = 0 or /soap:Header/wsse:Security/@soap:mustUnderstand = 0</Expression>
                 </VerifyContent>



                 <VerifyContent verificationAdapter="xpathadapter" description="'wsse:Security' elemani bir 'wsse:UsernameToken' elemani
icermelidir.">
                    <Expression>$soapHeader</Expression>
                    <Expression namespaceBindings = "{soap = http://schemas.xmlsoap.org/soap/envelope/, wsse = http://docs.oasis-open.org/
wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd, wsu = http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd}">
count(/soap:Header/wsse:Security/wsse:UsernameToken) = 1</Expression>
                 </VerifyContent>
                 <VerifyContent verificationAdapter="xpathadapter" description="'wsse:UsernameToken' elemani bir tane 'wsse:Username' elemani
 icermelidir">
                    <Expression>$soapHeader</Expression>
                    <Expression namespaceBindings = "{soap = http://schemas.xmlsoap.org/soap/envelope/, wsse = http://docs.oasis-open.org/
wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd, wsu = http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd}">
count(/soap:Header/wsse:Security/wsse:UsernameToken/wsse:Username) = 1</Expression>
                 </VerifyContent>

                 <VerifyContent verificationAdapter="xpathadapter" description="'wsse:UsernameToken' elemani bir tane 'wsse:Password' elemani
 icermelidir, ve bu elemanin 'Type' alani 'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText'
olmalidir.">
                    <Expression>$soapHeader</Expression>
                    <Expression namespaceBindings = "{soap = http://schemas.xmlsoap.org/soap/envelope/, wsse = http://docs.oasis-open.org/wss/
2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd, wsu = http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd}">
count(/soap:Header/wsse:Security/wsse:UsernameToken/wsse:Password[@Type = 'http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-username-token-profile-1.0#PasswordText']) = 1</Expression>
                 </VerifyContent>
                 <!--VerifyContent verificationAdapter="xpathadapter" description="'wsse:UsernameToken' elemani bir tane wsu:Created elemani
 icermelidir.">
                 <Expression>$soapHeader</Expression>
                 <Expression namespaceBindings = "{soap = http://schemas.xmlsoap.org/soap/envelope/, wsse = http://docs.oasis-open.org/wss/
2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd, wsu = http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd}">
count(/soap:Header/wsse:Security/wsse:UsernameToken/wsu:Created) = 1</Expression>
              </VerifyContent-->

              <VerifyContent verificationAdapter="xpathadapter" description="'wsse:UsernameToken' icin kullandiginiz  kullanici adi size atanan
  kullanici adi degerine esit olmalidir">
                 <Expression>$soapHeader</Expression>
                 <Expression namespaceBindings = "{soap = http://schemas.xmlsoap.org/soap/envelope/, wsse = http://docs.oasis-open.org/wss/2004/
01/oasis-200401-wss-wssecurity-secext-1.0.xsd, wsu = http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd}">
/soap:Header/wsse:Security/wsse:UsernameToken/wsse:Username/text() = $KullaniciAdi</Expression>
              </VerifyContent>
              <VerifyContent verificationAdapter="xpathadapter" description="'wsse:UsernameToken' icin kullandiginiz sifre size atanan sifre
degerine esit olmalidir">
                 <Expression>$soapHeader</Expression>
                 <Expression namespaceBindings = "{soap = http://schemas.xmlsoap.org/soap/envelope/, wsse = http://docs.oasis-open.org/wss/
2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd, wsu = http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd}">
/soap:Header/wsse:Security/wsse:UsernameToken/wsse:Password/text() = $Sifre</Expression>
              </VerifyContent>
              <!-- Set the error details of the message-->
              <WhenFalse>
                 <ThreadGroupRef nameRef = "TG03"/>
              </WhenFalse>
           </TestAssertion>
        </Thread>
     </ThreadGroup>

     <ThreadGroup name="TG03" description="WS-Username/Token profiline uymayan mesajlar icin hatalarin belirlenmesi">
        <Thread name = "TH03" description="WS-Username/Token profiline uymayan mesajlar icin hatalarin belirlenmesi">
           <Assign id="TH03_ASGN_01" variableRef="ackDetailCode" append="no_append">
              <Expression>'002-001'</Expression>
           </Assign>
           <Assign id="TH03_ASGN_02" variableRef="ackDetailDisplayName" append="no_append">
              <Expression>'Kullanici Adi ve Sifre Bilgileri Dogrulanamadi.'</Expression>
           </Assign>
           <Assign id="TH03_ASGN_03" variableRef="ackTypeCode" append="no_append">
              <Expression>'CE'</Expression>
           </Assign>
        </Thread>
     </ThreadGroup>

     <ThreadGroup name="TG05" description="Sozdizimsel hatalari gosteren mesajin olusturulmasi">
        <Thread name = "TH05" description="Sozdizimsel hatalari gosteren mesajin olusturulmasi">
           <Assign id="TH05_ASGN_01" variableRef="ackDetailCode" append="no_append">
              <Expression>'002-001'</Expression>
           </Assign>
           <Assign id="TH05_ASGN_02" variableRef="ackDetailDisplayName" append="no_append">
```

```
            <Expression>'Mesaj sozdizimsel olarak hatali'</Expression>
         </Assign>
         <Assign id="TH05_ASGN_03" variableRef="ackTypeCode" append="no_append">
            <Expression>'CE'</Expression>
         </Assign>
      </Thread>
   </ThreadGroup>

   <ThreadGroup name="TG06" description="Bildirim icin tasdik(acknowledgement) mesajinin donulmesi">
      <Thread name = "TH06" description="Bildirim icin tasdik(acknowledgement) mesajinin donulmesi">
         <!-- Set some parameters for the message that are unique or special-->
         <Assign id="TH06_ASGN_01" variableRef="messageTime" append="no_append">
            <Expression>srdc:getCurrentTimeWithFormat('yyyyMMddHHmmss')</Expression>
         </Assign>
         <Assign id="TH06_ASGN_02" variableRef="messageUUID" append="no_append">
            <Expression>srdc:generateUUID()</Expression>
         </Assign>
         <Assign id="TH06_ASGN_03" variableRef="messageID" append="no_append">
            <Expression>$MSVSbildirimi/MCCI_IN000001TR01/id/@extension</Expression>
         </Assign>
         <Assign id="TH06_ASGN_04" variableRef="senderID" append="no_append">
            <Expression>$MSVSbildirimi/MCCI_IN000001TR01/sender/device/id/@extension</Expression>
         </Assign>

         <SendMessage cpaID="Etkilesim1" id="S01" description="USBS, Uygulama Yanitini doner." transactionID="T01" toActor="USBSIstemci">
            <Transport transportHandler="HTTPSender"/>
            <Packaging packagingHandler="SOAPSender"/>
            <SetMessage description="Acknowledgement">
               <Content>
                  <Reference isDynamicContent = "true" mime_type="text/xml" href="testsuites/MOH_Muayene/MoHServer/msgtemplates/
MSVSAck_Generic.xml"/>
               </Content>
            </SetMessage>
         </SendMessage>
         <EndTransaction tid="T01"/>
      </Thread>
   </ThreadGroup>
   <ThreadGroupRef nameRef = "TG01"/>
</TestCase>
```

C.3 A Schematron Used in the Test Case (by Schematron Validator) for Validating Business Rules of Muayene Data Set

```
<?xml version="1.0" ?>
<sch:schema xmlns="http://www.ascc.net/xml/schematron"
   xmlns:sch="http://www.ascc.net/xml/schematron"
   xmlns:hl7 ="urn:hl7-org:v3">
   <sch:title>Muayene MSVS Veri seti Gonderim Kurallari</sch:title>
   <!--  assumes that effectiveTime value has at least 8 digits -->
   <sch:ns prefix="hl7" uri="urn:hl7-org:v3" />
   <sch:ns prefix="srdc" uri="java:com.srdc.testframework.testengine.expr.SRDCDateFunctions" />
   <sch:pattern name="rule 1 validation">
      <sch:rule context="hl7:reportSection//hl7:report/hl7:effectiveTime">
         <sch:assert test="root()//hl7:reportSection//hl7:report/hl7:code/@code"
            >Muayene MSVS veri seti: Kural 1 gecerli degil (Rapor turu alani, rapor tarihi alani oldugunda null olmamalidir):
            Rapor tarihi alani mevcut. Rapor turu null olmamali
         </sch:assert>
         <sch:report test="root()//hl7:reportSection//hl7:report/hl7:code/@code"
            >Muayene MSVS veri seti: Kural 1 gecerli (Rapor turu alani, rapor tarihi alani oldugunda null olmamalidir)
         </sch:report>
      </sch:rule>
   </sch:pattern>
   <sch:pattern name="rule 2 validation">
      <sch:rule context="hl7:reportSection//hl7:report/hl7:effectiveTime">
         <sch:assert test="
            srdc:isBeforeOrEqual( string(@value), srdc:formatXSDDate(string(current-dateTime()), 'yyyyMMddHHmmssZ') )
            ">Muayene MSVS veri seti: Kural 2 gecerli degil (Rapor tarihi alani, sistem tarihinden kucuk veya sistem tarihine esit olmalidir):
            Rapor Tarihi:<value-of select="@value"/> - Sistem Tarihi:<value-of select="current-dateTime()"/>
         </sch:assert>
         <sch:report test="
            srdc:isBeforeOrEqual( string(@value), srdc:formatXSDDate(string(current-dateTime()), 'yyyyMMddHHmmssZ') )
            ">Muayene MSVS veri seti: Kural 2 gecerli (Rapor tarihi alani, sistem tarihinden kucuk veya sistem tarihine esit olmalidir)
         </sch:report>
      </sch:rule>
   </sch:pattern>
   <sch:pattern name="rule 3 validation">
      <sch:rule context="hl7:reportSection//hl7:report/hl7:effectiveTime">
         <sch:assert test="
            srdc:isAfterOrEqual(string(@value), string(root()//hl7:examinationSection//hl7:encounter/hl7:effectiveTime/hl7:low/@value))
            ">Muayene MSVS veri seti: Kural 3 gecerli degil (Rapor tarihi alani, muayene baslangic tarihinden buyuk ya da o tarihe esit olmalidir):
            Rapor Tarihi:<value-of select="@value"/> - Muayane Baslangic Tarihi:<value-of select="root()//hl7:examinationSection//hl7:encounter/
hl7:effectiveTime/hl7:low/@value"/>
         </sch:assert>
         <sch:report test="
            srdc:isAfterOrEqual(string(@value), string(root()//hl7:examinationSection//hl7:encounter/hl7:effectiveTime/hl7:low/@value))
            ">Muayene MSVS veri seti: Kural 3 gecerli (Rapor tarihi alani, muayene baslangic tarihinden buyuk ya da o tarihe esit olmalidir)
         </sch:report>
      </sch:rule>
   </sch:pattern>
   <sch:pattern name="rule 4 validation">
      <sch:rule context="hl7:reportSection//hl7:report/hl7:effectiveTime">
         <sch:assert test="
            srdc:isAfterOrEqual(string(@value), string(root()//hl7:examinationSection//hl7:encounter/hl7:effectiveTime/hl7:high/@value))
            ">Muayene MSVS veri seti: Kural 4 gecerli degil (Rapor tarihi alani, muayene bitis tarihinden buyuk ya da o tarihe esit olmalidir):
            Rapor tarihi:<value-of select="@value"/> - Muayane Bitis Tarihi:<value-of select="root()//hl7:examinationSection//hl7:encounter/
hl7:effectiveTime/hl7:high/@value"/>
         </sch:assert>
         <sch:report test="
            srdc:isAfterOrEqual(string(@value), string(root()//hl7:examinationSection//hl7:encounter/hl7:effectiveTime/hl7:high/@value))
            ">Muayene MSVS veri seti: Kural 4 gecerli (Rapor tarihi alani, muayene bitis tarihinden buyuk ya da o tarihe esit olmalidir)
         </sch:report>
      </sch:rule>
   </sch:pattern>
   <sch:pattern name="rule 4 validation">
      <sch:rule context="hl7:examinationDataset//hl7:examProtocolNoSection//hl7:examProtocolNo/hl7:id">
         <sch:assert test="string-length(@extension) &lt; 21">Alan Kontrolu (Protokol Numarasi) gecerli degil (Protokol Numarasi alaninin
alan buyuklugu A(20) olmalidir):
            Protokol Numarasi:<value-of select="@extension"/>
         </sch:assert>
         <sch:report test="string-length(@extension) &lt; 21">Alan Kontrolu (Protokol Numarasi) gecerli (Protokol Numarasi alaninin alan
buyuklugu A(20) olmalidir)
         </sch:report>
      </sch:rule>
   </sch:pattern>
   <sch:pattern name="rule 4 validation">
      <sch:rule context="hl7:examinationDataset//hl7:investigationSection/hl7:text">
         <sch:assert test="string-length(text()) &lt; 501">Alan Kontrolu (Bulgu) gecerli degil (Bulgu alaninin alan buyuklugu A(500) olmalidir):
            Bulgu:<value-of select="text()"/>
         </sch:assert>
         <sch:report test="string-length(text()) &lt; 501">Alan Kontrolu (Bulgu) gecerli (Bulgu alaninin alan buyuklugu A(500) olmalidir)
         </sch:report>
      </sch:rule>
   </sch:pattern>
   <sch:pattern name="rule 4 validation">
      <sch:rule context="hl7:examinationDataset//hl7:historySection/hl7:text">
         <sch:assert test="string-length(text()) &lt; 501">Alan Kontrolu (Hikayesi) gecerli degil (Hikayesi alaninin alan buyuklugu A(500) olmalidir):
            Hikayesi:<value-of select="text()"/>
         </sch:assert>
         <sch:report test="string-length(text()) &lt; 501">Alan Kontrolu (Hikayesi) gecerli (Hikayesi alaninin alan buyuklugu A(500) olmalidir)
         </sch:report>
      </sch:rule>
   </sch:pattern>
   <sch:pattern name="rule 4 validation">
      <sch:rule context="hl7:examinationDataset//hl7:complaintSection/hl7:text">
         <sch:assert test="string-length(text()) &lt; 501">Alan Kontrolu (Sikayeti) gecerli degil (Sikayeti alaninin alan buyuklugu A(500) olmalidir):
            Sikayeti:<value-of select="text()"/>
         </sch:assert>
         <sch:report test="string-length(text()) &lt; 501">Alan Kontrolu (Sikayeti) gecerli (Sikayeti alaninin alan buyuklugu A(500) olmalidir)
         </sch:report>
      </sch:rule>
   </sch:pattern>
   <sch:pattern name="rule 5 validation">
      <sch:rule context="hl7:examinationDataset//hl7:diagnosisSection//hl7:diagnosis[./hl7:code/@code='EKTANI']">
         <sch:assert test="(not(./hl7:value/@code = ../preceding-sibling::*/hl7:diagnosis[./hl7:code/@code='EKTANI']/hl7:value/@code))">
            Muayene MSVS veri seti: Kural 5 gecerli degil (Ek tani alani ayni veri seti icinde ayni degerle birden fazla veri gonderilemez.)
            Ek tani alani degeri, kodu = <value-of select="./hl7:value/@code"/>
         </sch:assert>
         <sch:report test="(not(./hl7:value/@code = ../preceding-sibling::*/hl7:diagnosis[./hl7:code/@code='EKTANI']/hl7:value/@code)) and
(not(./hl7:value/@code = ../following-sibling::*/hl7:diagnosis[./hl7:code/@code='EKTANI']/hl7:value/@code))">
            Muayene MSVS veri seti: Kural 5 gecerli (Ek tani alani ayni veri seti icinde ayni degerle birden fazla veri gonderilemez.)
            Ek tani alani degeri, kodu = <value-of select="./hl7:value/@code"/>
         </sch:report>
      </sch:rule>
   </sch:pattern>
   <sch:pattern name="rule 6 validation">
      <sch:rule context="hl7:examinationDataset//hl7:reportSection//hl7:report">
         <sch:assert test="(not(./hl7:code/@code = ../preceding-sibling::*/hl7:report/hl7:code/@code))">
            Muayene MSVS veri seti: Kural 6 gecerli degil (Rapor Turu alani ayni veri seti icinde ayni degerle birden fazla veri gonderilemez.)
            Rapor Turu degeri, kodu = <value-of select="./hl7:code/@code"/>
         </sch:assert>
         <sch:report test="(not(./hl7:code/@code = ../preceding-sibling::*/hl7:report/hl7:code/@code)) and (not(./hl7:code/@code = ../
following-sibling::*/hl7:report/hl7:code/@code))">
            Muayene MSVS veri seti: Kural 6 gecerli (Rapor Turu alani ayni veri seti icinde ayni degerle birden fazla veri gonderilemez.)
            Rapor Turu degeri, kodu = <value-of select="./hl7:code/@code"/>
         </sch:report>
      </sch:rule>
   </sch:pattern>
</sch:schema>
```

C.4 A Message Template used in the Test Case

```
<?xml version='1.0' encoding='UTF-8'?>
<MCCI_IN000002TR01 xmlns="urn:hl7-org:v3" xmlns:xlink="http://www.w3.org/TR/WD-xlink" xmlns:mif="urn:hl7-org:v3/mif"
   xmlns:ex="urn:hl7-org/v3-example" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <id root="2.16.840.1.113883.3.129.2.1.2" extension="%$messageUUID%" />
   <creationTime value="%$messageTime%" />
   <interactionId root="2.16.840.1.113883.3.129.2.1.1" extension="MCCI_IN000002TR01" />
   <processingCode code="T" />
   <processingModeCode code="T" />
   <acceptAckCode code="NE" />
   <receiver typeCode="RCV">
      <device classCode="DEV" determinerCode="INSTANCE">
         <id root="2.16.840.1.113883.3.129.1.1.5" extension="%$senderID%" />
      </device>
   </receiver>
   <sender typeCode="SND">
      <device classCode="DEV" determinerCode="INSTANCE">
         <id root="2.16.840.1.113883.3.129.1.1.5" extension="USBS" />
      </device>
   </sender>
   <acknowledgement>
      <typeCode code="%$ackTypeCode%" />
      <message>
         <id root="2.16.840.1.113883.3.129.2.1.2" extension="%$messageID%" />
      </message>
   </acknowledgement>
</MCCI_IN000002TR01>
```

APPENDIX D

TEST AND SUITES ADAPTER METADATA FOR IHE SWF CASE STUDY

D.1 Metadata of the DICOMReceiver

```
<MessagingHandler
   name="DICOMReceiver"
   complementaryMessagingHandler="DICOMSender"
   layer="TransportLayer"
   role="Receiver"
   version="1.0"
   author="SRDC-Team"
   class="com.srdc.testframework.dicom.transport.DICOMReceiver">
   <ExpectedConfigurationAttributes>
      <tss:ConfigurationAttribute name="name">TestBATN Engine</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="aeTitle">TestBATN</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="dimseRspTimeout">10000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="maxPDULengthReceive">16384</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="maxPDULengthSend">16384</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="connectTimeout">5000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="acceptTimeout">5000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="requestTimeout">5000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="releaseTimeout">5000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="socketCloseDelay">50</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="sendBufferSize">0</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="receiveBufferSize">0</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="tcpdelay">true</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="idleTimeout">60000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="retrieveRspTimeout">600000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="isExpectingRelease">false</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="isExpectingAbort">false</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="presentationContextID" isRequired="true">null</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="releaseWhenFinished">false</tss:ConfigurationAttribute>
   </ExpectedConfigurationAttributes>
   <InputArguments/>
   <OutputArguments>
      <Argument required="true" name="associate-rq" description="A-ASSOCIATE-RQ message recevied from SUT">
 <ApplicableType>object</ApplicableType>
      </Argument>
      <Argument required="true" name="associate-ac" description="A-ASSOCIATE-AC or A-ASSOCIATE-RJ message that has sent to the SUT">
 <ApplicableType>object</ApplicableType>
      </Argument>
      <Argument required="true" name="pdv-command" description="PDV Command part of the message that has received from the SUT">
 <ApplicableType>dicom</ApplicableType>
```

```
      </Argument>
      <Argument required="true" name="pdv-dataset" description="PDV Dataset part of the message that has received from the SUT">
 <ApplicableType>dicom</ApplicableType>
      </Argument>
      <Argument required="true" name="a-abort" description="A-Abort message that has received from the SUT, if abort is expected">
 <ApplicableType>object</ApplicableType>
      </Argument>
   </OutputArguments>
</MessagingHandler>
```

D.2 Metadata of the DICOMSender

```
<MessagingHandler
   name="DICOMSender"
   layer="TransportLayer"
   role="Sender"
   version="1.0"
   author="SRDC-Team"
   class="com.srdc.testframework.dicom.transport.DICOMSender">

   <ExpectedConfigurationAttributes>
      <tss:ConfigurationAttribute name="name">TestBATN Engine</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="aeTitle">TestBATN</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="offerDefaultTSInSeparatePC">false</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name= "username">null</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name= "password">null</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name= "useridposresp">false</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name= "packPDV">true</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name= "reaperPeriod">10000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="dimseRspTimeout">10000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="maxPDULengthReceive">16384</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="maxPDULengthSend">16384</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="connectTimeout">5000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="acceptTimeout">5000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="requestTimeout">5000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="releaseTimeout">5000</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="socketCloseDelay">50</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="sendBufferSize">0</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="receiveBufferSize">0</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="tcpdelay">true</tss:ConfigurationAttribute>

      <tss:ConfigurationAttribute name="isSendRelease">false</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="isSendAbort">false</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="presentationContextID" isRequired="true">null</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</tss:ConfigurationAttribute>
      <tss:ConfigurationAttribute name= "remoteAETitle" isRequired="true">null</tss:ConfigurationAttribute>
   </ExpectedConfigurationAttributes>
   <InputArguments>
      <Argument required="true" name="as-name" description="Abstract syntax name (SOP Class UID) for the message to be send">
 <ApplicableType>string</ApplicableType>
      </Argument>
      <Argument required="true" name="ts-name" description="Transfer syntax id for the message to be send">
 <ApplicableType>string</ApplicableType>
      </Argument>
      <Argument required="false" name="pdv-command" description="PDV(DIMSE) Command part of the message to be send">
 <ApplicableType>dicom</ApplicableType>
      </Argument>
      <Argument required="false" name="pdv-dataset" description="PDV(DIMSE) Dataset part of the message to be send">
 <ApplicableType>dicom</ApplicableType>
      </Argument>
   </InputArguments>
   <OutputArguments/>
</MessagingHandler>
```

D.3 Metadata of the DIMSESender

```
<MessagingHandler
   name="DIMSESender"
   layer="PackagingLayer"
   role="Sender"
   version="1.0"
   author="SRDC-Team"
   class="com.srdc.testframework.dicom.transport.DIMSESender">
   <ExpectedConfigurationAttributes>
      <tss:ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</tss:ConfigurationAttribute>
   </ExpectedConfigurationAttributes>
   <InputArguments>
      <!--
 *        DIMSE Command should be one of the followings
 C_STORE_RQ
 C_STORE_RSP
 C_GET_RQ
 C_GET_RSP
 C_FIND_RQ
 C_FIND_RSP
 C_MOVE_RQ
 C_MOVE_RSP
 C_ECHO_RQ
 C_ECHO_RSP
 N_EVENT_REPORT_RQ
 N_EVENT_REPORT_RSP
 N_GET_RQ
 N_GET_RSP
 N_SET_RQ
 N_SET_RSP
 N_ACTION_RQ
 N_ACTION_RSP
 N_CREATE_RQ
 N_CREATE_RSP
 N_DELETE_RQ
 N_DELETE_RSP
 C_CANCEL_RQ
 *
 -->
 <Argument required="true" name="dimse-cmd" description="DIMSE COMMAND NAME">
    <ApplicableType>string</ApplicableType>
 </Argument>
 <Argument required="true" name="dimse-dataset" description="DIMSE Dataset">
    <ApplicableType>dicom</ApplicableType>
 </Argument>
 <Argument required="true" name="msgId" description="MessageId">
    <ApplicableType>string</ApplicableType>
 </Argument>
 <Argument required="true" name="cuid" description="Affected SOP Class UID">
    <ApplicableType>string</ApplicableType>
 </Argument>
 <Argument required="false" name="optional1" description="Affected SOP Instance UID or Priority or Status according to command">
    <ApplicableType>string</ApplicableType>
 </Argument>
 <Argument required="false" name="optional2" description="Priority or Destination or eventTypeID or Tags or actionTypeID">
    <ApplicableType>string</ApplicableType>
    <ApplicableType>list_string</ApplicableType>
 </Argument>
 <Argument required="false" name="moveOriginatorAET" description="MoveOriginator AET for C_STORE_RQ">
    <ApplicableType>string</ApplicableType>
 </Argument>
 <Argument required="false" name="moveOriginatorMsgId" description="moveOriginatorMsgId for C_STORE_RQ">
    <ApplicableType>string</ApplicableType>
 </Argument>
      </InputArguments>
      <OutputArguments/>
</MessagingHandler>
```

D.4 Metadata of the DICOMValidator

```
<VerificationAdapter
   name="dicomadapter"
   version="0.9"
   author="SRDC-Team"
   verificationFunctionRequired="false"
   class="com.srdc.testframework.dicom.validator.DICOMValidator">
   <AdapterAttribute name="sopClassUID" type="string" description="SOP Class UID"/>
   <AdapterAttribute name="dimseCommandName" type="string" description="DIMSE Command Name"/>
   <AdapterAttribute name="command" type="dicom" description="DIMSE Command"/>
   <AdapterAttribute name="dataset" type="dicom" description="DIMSA Dataset"/>
</VerificationAdapter>
```

D.5 Test Case Definition for IHE 2011 Connectathon SWF-OF-Modality Interface Scenario

```
<?xml version="1.0" encoding="UTF-8"?>
<TestCase xmlns="model.testsuite.testframework.srdc.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   id = "SWF_OF_Modality_Interface" description = "IHE 2011 Connectathon test scenario, SWF_OF_Modality, for OF_Modality Interactions">
   <MetaData>
      <Title>IHE 2011 Connectathon test scenario, SWF_OF_Modality, for OF_Modality Interactions</Title>
      <Version>0.1</Version>
      <Maintainer>SRDC Ltd.</Maintainer>
      <Location>Ankara, Turkey</Location>
      <PublishDate>17/03/2011</PublishDate>
      <Status>IN USE</Status>
      <Description>This test scenario is for testing the interface of DSS/OF actor in the sub workflow among ADT, Order Placer,
Modality and DSS/OF actors in IHE SWF profile.
      </Description>
   </MetaData>
   <!-- Parties Participating the Test-->
   <ActorUnderTest>ADT_OP_MODALITY_IM</ActorUnderTest>
   <!-- The system under test is playing the DSS/Order Filler role -->
   <ActorUnderTest>DSSOF</ActorUnderTest>
   <!--TestBATN is playing Patient Registration (ADT), Order Placer(OP), Image Manager(IM) and Modality (imaging device) roles-->
   <TestCaseCoordinator>ADT_OP_MODALITY_IM</TestCaseCoordinator>

   <ConfigurationGroup>
      <CPA id="HL7_INT1" fromActor="ADT_OP_MODALITY_IM" toActor="DSSOF">
 <CPAType>Interaction for RAD1 Transaction between ADT-DSSOF and RAD2 Transaction between OP-DSSOF</CPAType>
 <CPAId>http://www.srdc.com.tr/cpa1</CPAId>
      </CPA>
      <CPA id="HL7_INT2" fromActor="DSSOF" toActor="ADT_OP_MODALITY_IM">
 <CPAType>Interaction for RAD4 Transaction between DSSOF-IM</CPAType>
 <CPAId>http://www.srdc.com.tr/cpa2</CPAId>
      </CPA>
```

```
     <CPA id="DICOM_INT1" fromActor="ADT_OP_MODALITY_IM" toActor="DSSOF">
<CPAType>Interaction for RAD5 Transaction between Modality-DSSOF</CPAType>
<CPAId>http://www.srdc.com.tr/cpa3</CPAId>
     </CPA>
     <CPA id="DICOM_INT2" fromActor="ADT_OP_MODALITY_IM" toActor="DSSOF">
<CPAType>Interaction for RAD6 and RAD7 Transactions between Modality-DSSOF</CPAType>
<CPAId>http://www.srdc.com.tr/cpa4</CPAId>
     </CPA>
  </ConfigurationGroup>

  <Variables>
     <Variable name="hl7Acknowledgement" description="HL7 Acknowledgement">
<Type>xml</Type>
     </Variable>
     <Variable name="hl7Message2" description="HL7 Message">
<Type>xml</Type>
     </Variable>
     <Variable name="hl7Message" description="HL7 Message">
<Type>xml</Type>
     </Variable>
     <Variable name="messageControlIdentifer" description="MSH.10 messageControlID">
<Type>string</Type>
     </Variable>

     <!-- DICOM variables-->
     <Variable name="assoc_rq" description="A-Associate-RQ object">
<Type>object</Type>
     </Variable>
     <Variable name="assoc_ac" description="A-Associate-AC or A-Associate-RJ object">
<Type>object</Type>
     </Variable>
     <Variable name="pdv_command" description="Dicom PDV Command(DIMSE Command)">
<Type>dicom</Type>
     </Variable>
     <Variable name="pdv_dataset" description="Dicom PDV Dataset(DIMSE Dataset)">
<Type>dicom</Type>
     </Variable>
     <Variable name="a_abort" description="A-Abort object">
<Type>object</Type>
     </Variable>
     <Variable name="n-set-rq" description="Dimse Command">
<Type>string</Type>
<Value>N-SET-RQ</Value>
     </Variable>
     <Variable name="n-set-rsp" description="Dimse Command">
<Type>string</Type>
<Value>N-SET-RSP</Value>
     </Variable>
     <Variable name="n-create-rq" description="Dimse Command">
<Type>string</Type>
<Value>N-CREATE-RQ</Value>
     </Variable>
     <Variable name="n-create-rsp" description="Dimse Command">
<Type>string</Type>
<Value>N-CREATE-RSP</Value>
     </Variable>
     <Variable name="dimse-cmd-req" description="Dimse Command">
<Type>string</Type>
<Value>C-FIND-RQ</Value>
     </Variable>
     <Variable name="dimse-cmd" description="Dimse Command">
<Type>string</Type>
<Value>C-FIND-RSP</Value>
```

```
     </Variable>
     <Variable name="msgId" description="Dimse Message Id">
<Type>string</Type>
<Value>1387</Value>
     </Variable>
     <Variable name="MPPSCUID" description="Service Class Id">
<Type>string</Type>
<Value>1.2.840.10008.3.1.2.3.3</Value>
     </Variable>
     <Variable name="cuid" description="Service Class Id">
<Type>string</Type>
<Value>1.2.840.10008.5.1.4.31</Value>
     </Variable>
     <Variable name="statusok" description="Status OK">
<Type>string</Type>
<Value>1</Value>
     </Variable>
     <Variable name="iuid" description="Affected SOAP Instance UID">
<Type>string</Type>
<Value>1.3.12.2.1107.5.2.5.11090.5.0.124643</Value>
     </Variable>
     <Variable name="iuid2" description="Affected SOAP Instance UID">
<Type>string</Type>
<Value>1.3.12.2.1107.5.2.5.11090.5.0.146643</Value>
     </Variable>
     <Variable name="priority" description="priority">
<Type>string</Type>
<Value>1</Value>
     </Variable>
     <Variable name="cfind_dataset_xml" description="Dimse Dataset">
<Type>xml</Type>
     </Variable>
     <Variable name="cfind_dataset_dcm" description="Dimse Dataset">
<Type>dicom</Type>
     </Variable>
     <Variable name="cfind_rsp_xml" description="Dimse Dataset">
<Type>xml</Type>
     </Variable>
     <Variable name="cfind_rsp_xml_dataset" description="Dimse Dataset">
<Type>xml</Type>
     </Variable>
     <Variable name="numofmatch" description="Number of CFIND matches">
<Type>string</Type>
<Value>0</Value>
     </Variable>
     <Variable name="studyInstanceUID" description="Study UID">
<Type>string</Type>
     </Variable>
     <Variable name="studyID" description="Study UID">
<Type>string</Type>
     </Variable>
     <Variable name="rpID" description="Study UID">
<Type>string</Type>
     </Variable>
     <Variable name="spsID" description="Study UID">
<Type>string</Type>
     </Variable>
     <Variable name="accNumber" description="Study UID">
<Type>string</Type>
     </Variable>
     <Variable name="orderFillerNR" description="Study UID">
<Type>string</Type>
     </Variable>
     <Variable name="orderFillerAuth" description="Study UID">
<Type>string</Type>
     </Variable>
     <Variable name="patientName" description="Number of CFIND matches">
<Type>string</Type>
<Value>DURSUN^AHMET*</Value>
     </Variable>
     <Variable name="dummy" description="Study UID">
<Type>string</Type>
     </Variable>
  </Variables>

  <ThreadGroup name="TG02" description="Loop for C-FIND responses">
     <Thread name = "TH02" description="Loop for C-FIND responses">
<ReceiveMessage id="R4" transactionID="RAD5" fromActor="DSSOF" description = "Please send the responses for the RAD5 Modality Worklist Query (C-FIND RSP)">
   <Transport transportHandler="DICOMReceiver">
      <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
      <ConfigurationAttribute name="presentationContextID">1.2.840.10008.5.1.4.31</ConfigurationAttribute>
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
      <ConfigurationAttribute name="releaseWhenFinished">false</ConfigurationAttribute>
   </Transport>
   <VariableRef>assoc_rq</VariableRef>
   <VariableRef>assoc_ac</VariableRef>
   <VariableRef>pdv_command</VariableRef>
   <VariableRef>pdv_dataset</VariableRef>
   <VariableRef>a_abort</VariableRef>
</ReceiveMessage>
<Assign id="ASGN_1" variableRef="cfind_rsp_xml" append="no_append">
   <Expression>$pdv_command</Expression>
</Assign>
<TestAssertion id="TA4" description = "Check if status is pending">
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check the CFIND status">
      <Expression>$cfind_rsp_xml</Expression>
      <Expression>/dicom/attr[@tag='00000900']/text() = '65280' or /dicom/attr[@tag='00000900']/text() = '65281'</Expression>
   </VerifyContent>
   <WhenTrue>
      <ThreadGroupRef nameRef = "TG03"/>
   </WhenTrue>
</TestAssertion>
     </Thread>
  </ThreadGroup>

  <ThreadGroup name="TG03" description="Validation of C-FIND responses">
     <Thread name = "TH03" description="Validation of C-FIND responses">
<Assign id="ASGN_2" variableRef="numofmatch" append="no_append">
   <Expression namespaceBindings="{srdc = http://srdc.testframework.default}">srdc:add($numofmatch, '1')</Expression>
</Assign>
<Assign id="ASGN_3" variableRef="cfind_rsp_xml_dataset" append="no_append">
   <Expression>$pdv_dataset</Expression>
</Assign>
<Assign id="ASGN_3a" variableRef="studyInstanceUID" append="no_append">
   <Expression>$cfind_rsp_xml_dataset/dicom/attr[@tag='0020000D']/text()</Expression>
</Assign>
<Assign id="ASGN_3c" variableRef="rpID" append="no_append">
   <Expression>$cfind_rsp_xml_dataset/dicom//attr[@tag='00401001']/text()</Expression>
</Assign>
<Assign id="ASGN_3d" variableRef="spsID" append="no_append">
   <Expression>$cfind_rsp_xml_dataset/dicom//attr[@tag='00400009']/text()</Expression>
</Assign>
<Assign id="ASGN_3e" variableRef="accNumber" append="no_append">
   <Expression>$cfind_rsp_xml_dataset/dicom//attr[@tag='00080050']/text()</Expression>
</Assign>
```

```
<TestAssertion id="TA5" description = "Validate DICOM Content according to Table F.7.2-1">
   <VerifyContent verificationAdapter="dicomadapter"
      description="Validate DICOM Content">
      <Expression>$cuid</Expression>
      <Expression>$dimse-cmd</Expression>
      <Expression>$pdv_command</Expression>
      <Expression>$pdv_dataset</Expression>
   </VerifyContent>
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check whether Image Manager returns response message id properly.">
      <Expression>$cfind_rsp_xml</Expression>
      <Expression>/dicom/attr[@tag='00000120']/text() = '1387'</Expression>
   </VerifyContent>
</TestAssertion>
<ThreadGroupRef nameRef = "TG02"/>
     </Thread>
  </ThreadGroup>

  <ThreadGroup name="TG01" description="Main ThreadGroup">
     <Thread name="TH01" description="Basic Steps">
<!-- RAD1 Patient Registration transaction from ADT to DSSOF-->
<BeginTransaction tid="RAD1" cpaID="HL7_INT1"/>
<SendMessage id="S01" description="Send an ADT-A01  Message" transactionID="RAD1" toActor="DSSOF">
   <Transport transportHandler="HL7MLLPSender">
   </Transport>
   <SetMessage description="ADT-A01_Message">
      <Content>
 <Reference isDynamicContent = "true" mime_type="text" href="testsuites/Dicom_IHEPIR/msgtemplates/SWF_OF_Modality_Interface/ADT-A01.txt"/>
      </Content>
   </SetMessage>
</SendMessage>
<ReceiveMessage id="R01" transactionID="RAD1" fromActor="DSSOF" description="Receive HL7 Acknowledgement.." >
   <Transport transportHandler="HL7MLLPReceiver">
   </Transport>
   <VariableRef>hl7Acknowledgement</VariableRef>
</ReceiveMessage>
<EndTransaction tid="RAD1"/>
<TestAssertion id="TA1" description = "Check according to message profile">
   <ValidateContent validationAdapter="hl7xmladapter"
      schemaLocation="testsuites/Dicom_IHEPIR/profiles/1.3.6.1.4.12559.11.1.1.21.xml"
      description="The control of the ACK message with the HL7 message profile.">
      <Expression>$hl7Acknowledgement</Expression>
   </ValidateContent>
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check whether Image Manager returns the sent messageControlID.">
      <Expression>$hl7Acknowledgement</Expression>
      <Expression>/ACK/MSA/MSA.2/text() = '1390'</Expression>
   </VerifyContent>
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check whether Image Manager returns ackCode AA.">
      <Expression>$hl7Acknowledgement</Expression>
      <Expression>/ACK/MSA/MSA.1/text() = 'AA'</Expression>
   </VerifyContent>
</TestAssertion>
<!-- RAD2 HL7 New Order transaction from OP to DSSOF-->
<BeginTransaction tid="RAD2" cpaID="HL7_INT1"/>
<SendMessage id="S0187" description="Send an ORM-O01  Message" transactionID="RAD2" toActor="DSSOF">
   <Transport transportHandler="HL7MLLPSender">
   </Transport>
   <SetMessage description="ORM-O01_Message">
      <Content>
 <Reference isDynamicContent = "true" mime_type="text" href="testsuites/Dicom_IHEPIR/msgtemplates/SWF_OF_Modality_Interface/ORM_O01_Case2.txt"/>
      </Content>
```

```
    </SetMessage>
 </SendMessage>
 <ReceiveMessage id="R08781" transactionID="RAD2" fromActor="DSSOF" description="Receive HL7 Acknowledgement.." >
    <Transport transportHandler="HL7MLLPReceiver">
    </Transport>
    <VariableRef>hl7Acknowledgement</VariableRef>
 </ReceiveMessage>
 <EndTransaction tid="RAD2"/>
 <TestAssertion id="TA78781" description = "Check according to message profile">
    <ValidateContent validationAdapter="hl7xmladapter"
       schemaLocation="testsuites/Dicom_Deneme/profiles/1.3.6.1.4.12559.11.1.1.21.xml"
       description="The control of the ACK message with the HL7 message profile.">
       <Expression>$hl7Acknowledgement</Expression>
    </ValidateContent>
    <VerifyContent verificationAdapter="xpathadapter"
       description="Check whether Image Manager returns the sent messageControlID.">
       <Expression>$hl7Acknowledgement</Expression>
       <Expression>/ACK/MSA/MSA.2/text() = '1390'</Expression>
    </VerifyContent>
    <VerifyContent verificationAdapter="xpathadapter"
       description="Check whether Image Manager returns ackCode AA.">
       <Expression>$hl7Acknowledgement</Expression>
       <Expression>/ACK/MSA/MSA.1/text() = 'AA'</Expression>
    </VerifyContent>
 </TestAssertion>
<!-- RAD4 Procedure Schedules Message From DSSOF to Imagae Manager-->
 <BeginTransaction tid="RAD4" cpaID="HL7_INT2"/>
 <ReceiveMessage id="R03" transactionID="RAD4" fromActor="DSSOF" description="Receive ORM-O01 message.." >
    <Transport transportHandler="HL7MLLPReceiver">
    </Transport>
    <VariableRef>hl7Message</VariableRef>
 </ReceiveMessage>
 <TestAssertion id="TA3" description = "Check according to message profile">
    <ValidateContent validationAdapter="hl7xmladapter"
       schemaLocation="testsuites/Dicom_IHEPIR/profiles/ORM-O01-Profile.xml"
       description="The control of the ACK message with the HL7 message profile.">
       <Expression>$hl7Message</Expression>
    </ValidateContent>
 </TestAssertion>
 <Assign id="ASGN_325" variableRef="messageControlIdentifer" append="no_append">
    <Expression>$hl7Message//MSH/MSH.10/text()</Expression>
 </Assign>
 <SendMessage id="S03" description="Send an ACK Message" transactionID="RAD4" toActor="DSSOF">
    <Transport transportHandler="HL7MLLPSender">
    </Transport>
    <SetMessage description="ADT_Message">
       <Content>
  <Reference isDynamicContent = "true" mime_type="text" href="testsuites/Dicom_IHEPIR/msgtemplates/SWF_OF_Modality_Interface/ADT-ACK.txt"/>
       </Content>
    </SetMessage>
 </SendMessage>
 <EndTransaction tid="RAD4"/>

 <AskTestData toActor="DSSOF" id="ATD1" description="Mola">
    <TestQuestion id="TQ1" question="..." description="...">
       <VariableRef>dummy</VariableRef>
    </TestQuestion>
 </AskTestData>
<!-- RAD5 Modality Worklist Query from Modality to DSSOF-->
 <BeginTransaction tid="RAD5" cpaID="DICOM_INT1"/>
 <Assign id="ASGN_4" variableRef="cfind_dataset_xml" append="no_append">
    <Expression>#testsuites/Dicom_IHEPIR/msgtemplates/SWF_OF_Modality_Interface/mwl-Case2.xml</Expression>
 </Assign>
```

```
<Assign id="ASGN_5" variableRef="cfind_dataset_dcm" append="no_append">
   <Expression>$cfind_dataset_xml</Expression>
</Assign>
<SendMessage id="S06" description="The Modality will send the RAD5-Modality Worklist Query(DICOM C-FIND) to your application" transactionID="RAD5" toActor="D
   <Transport transportHandler="DICOMSender">
      <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
      <ConfigurationAttribute name="presentationContextID">1.2.840.10008.5.1.4.31</ConfigurationAttribute>
      <ConfigurationAttribute name="remoteAETitle">DCM_SERVER</ConfigurationAttribute>
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
   </Transport>
   <Packaging packagingHandler="DIMSESender">
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
   </Packaging>
   <SetMessage description="C-FIND REQUEST">
      <Content><VariableRef>dimse-cmd-req</VariableRef></Content>
      <Content>
 <VariableRef>cfind_dataset_dcm</VariableRef>
      </Content>
      <Content><VariableRef>msgId</VariableRef></Content>
      <Content><VariableRef>cuid</VariableRef></Content>
      <Content><VariableRef>priority</VariableRef></Content>
   </SetMessage>
</SendMessage>
<ThreadGroupRef nameRef = "TG02"/>
<TestAssertion id="TA6" description = "Check if status is complete and we get a match for the query">
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check the CFIND status">
      <Expression>$cfind_rsp_xml</Expression>
      <Expression>/dicom/attr[@tag='00000900']/text() = '0'</Expression>
   </VerifyContent>
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check if the number of matches are as expected (1 match)">
      <Expression>null</Expression>
      <Expression>$numofmatch = 1</Expression>
   </VerifyContent>
</TestAssertion>
<SendMessage id="S07" description="TestBATN_SCP will send an A-RELEASE to your application" transactionID="RAD5" toActor="DSSOF">
   <Transport transportHandler="DICOMSender">
      <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
      <ConfigurationAttribute name="presentationContextID">1.2.840.10008.5.1.4.31</ConfigurationAttribute>
      <ConfigurationAttribute name="remoteAETitle">DCM_SERVER</ConfigurationAttribute>
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
      <ConfigurationAttribute name="isSendRelease">true</ConfigurationAttribute>
   </Transport>
   <SetMessage description="Nothing">
      <Content><NullVariable>null</NullVariable></Content>
      <Content><NullVariable>null</NullVariable></Content>
      <Content><NullVariable>null</NullVariable></Content>
      <Content><NullVariable>null</NullVariable></Content>
      <Content><NullVariable>null</NullVariable></Content>
   </SetMessage>
</SendMessage>
<EndTransaction tid="RAD5"/>
<!-- RAD6 Performed Procedure Step In Progress from Modality to DSSOF-->
<BeginTransaction tid="RAD6" cpaID="DICOM_INT2"/>
<Assign id="ASGN_6a" variableRef="cfind_dataset_xml" append="no_append">
   <Expression>#testsuites/Dicom_IHEPIR/msgtemplates/SWF_OF_Modality_Interface/mpps-ncreate-Case2.xml</Expression>
</Assign>
<Assign id="ASGN_7a" variableRef="cfind_dataset_dcm" append="no_append">
   <Expression>$cfind_dataset_xml</Expression>
</Assign>
<SendMessage id="S12" description="TestBATN_SCP will send an N_CREATE_REQUEST to your application" transactionID="RAD6" toActor="DSSOF">
   <Transport transportHandler="DICOMSender">
```

```
      <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
      <ConfigurationAttribute name="presentationContextID">1.2.840.10008.3.1.2.3.3</ConfigurationAttribute>
      <ConfigurationAttribute name="remoteAETitle">DCM_SERVER</ConfigurationAttribute>
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
   </Transport>
   <Packaging packagingHandler="DIMSESender">
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
   </Packaging>
   <SetMessage description="N_CREATE REQUEST">
      <Content><VariableRef>n-create-rq</VariableRef></Content>
      <Content>
 <VariableRef>cfind_dataset_dcm</VariableRef>
      </Content>
      <Content><VariableRef>msgId</VariableRef></Content>
      <Content><VariableRef>MPPSCUID</VariableRef></Content>
      <Content><VariableRef>iuid</VariableRef></Content>
   </SetMessage>
</SendMessage>

<!--RAD3 Order Status Update from DSSOF to OP-->
<BeginTransaction tid="RAD3b" cpaID="HL7_INT2"/>
<ReceiveMessage id="R03_b" transactionID="RAD3b" fromActor="DSSOF" description="Receive RAD-3 Order Status Update" >
   <Transport transportHandler="HL7MLLPReceiver">
   </Transport>
   <VariableRef>hl7Message</VariableRef>
</ReceiveMessage>
<TestAssertion id="TA3_b" description = "Check according to message profile">
   <ValidateContent validationAdapter="hl7xmladapter"
      schemaLocation="testsuites/Dicom_IHEPIR/profiles/ORM-O01-Profile.xml"
      description="The control of the ACK message with the HL7 message profile.">
      <Expression>$hl7Message</Expression>
   </ValidateContent>
</TestAssertion>
<Assign id="ASGN_32_b" variableRef="messageControlIdentifer" append="no_append">
   <Expression>$hl7Message//MSH/MSH.10/text()</Expression>
</Assign>
<SendMessage id="S03_b" description="Send an ACK Message" transactionID="RAD3b" toActor="DSSOF">
   <Transport transportHandler="HL7MLLPSender">
   </Transport>
   <SetMessage description="ADT_Message">
      <Content>
 <Reference isDynamicContent = "true" mime_type="text" href="testsuites/Dicom_IHEPIR/msgtemplates/RAD12/ADT-ACK.txt"/>
      </Content>
   </SetMessage>
</SendMessage>
<EndTransaction tid="RAD3b"/>
<!-- Response to RAD6 Performed Procedure Step In Progress from DSSOF to Modality-->
<ReceiveMessage id="R12" transactionID="RAD6" fromActor="DSSOF" description = "Please send the RAD 6 (NCREATE RSP MPPS)">
   <Transport transportHandler="DICOMReceiver">
      <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
      <ConfigurationAttribute name="presentationContextID">1.2.840.10008.3.1.2.3.3</ConfigurationAttribute>
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
      <ConfigurationAttribute name="releaseWhenFinished">true</ConfigurationAttribute>
   </Transport>
   <VariableRef>assoc_rq</VariableRef>
   <VariableRef>assoc_ac</VariableRef>
   <VariableRef>pdv_command</VariableRef>
   <VariableRef>pdv_dataset</VariableRef>
   <VariableRef>a_abort</VariableRef>
</ReceiveMessage>
<Assign id="ASGN_08a" variableRef="cfind_dataset_xml" append="no_append">
   <Expression>$pdv_command</Expression>
</Assign>
```

```
<TestAssertion id="TA12" description = "Validate DICOM Content according to Table F.7.2-1">
   <VerifyContent verificationAdapter="dicomadapter"
      description="Validate DICOM Content">
      <Expression>$MPPSCUID</Expression>
      <Expression>$n-create-rsp</Expression>
      <Expression>$pdv_command</Expression>
      <Expression>$pdv_dataset</Expression>
   </VerifyContent>
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check whether Image Manager returns ack code properly.">
      <Expression>$cfind_dataset_xml</Expression>
      <Expression>/dicom/attr[@tag='00000900']/text() = '0'</Expression>
   </VerifyContent>
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check whether Image Manager returns response message id properly.">
      <Expression>$cfind_dataset_xml</Expression>
      <Expression>/dicom/attr[@tag='00000120']/text() = '1387'</Expression>
   </VerifyContent>
</TestAssertion>
<EndTransaction tid="RAD6"/>
<!-- RAD7 Performed Procedure Step Completed from Modality to DSSOF-->
<BeginTransaction tid="RAD7" cpaID="DICOM_INT2"/>
<Assign id="ASGN_6" variableRef="cfind_dataset_xml" append="no_append">
   <Expression>#testsuites/Dicom_IHEPIR/msgtemplates/SWF_OF_Modality_Interface/mpps-nset-fixed-Case2.xml</Expression>
</Assign>
<Assign id="ASGN_7" variableRef="cfind_dataset_dcm" append="no_append">
   <Expression>$cfind_dataset_xml</Expression>
</Assign>
<SendMessage id="S08" description="TestBATN_SCP will send an N_SET_REQUEST to your application" transactionID="RAD7" toActor="DSSOF">
   <Transport transportHandler="DICOMSender">
      <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
      <ConfigurationAttribute name="presentationContextID">1.2.840.10008.3.1.2.3.3</ConfigurationAttribute>
      <ConfigurationAttribute name="remoteAETitle">DCM_SERVER</ConfigurationAttribute>
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
   </Transport>
   <Packaging packagingHandler="DIMSESender">
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
   </Packaging>
   <SetMessage description="N_SET REQUEST">
      <Content><VariableRef>n-set-rq</VariableRef></Content>
      <Content>
 <VariableRef>cfind_dataset_dcm</VariableRef>
      </Content>
      <Content><VariableRef>msgId</VariableRef></Content>
      <Content><VariableRef>MPPSCUID</VariableRef></Content>
      <Content><VariableRef>iuid</VariableRef></Content>
   </SetMessage>
</SendMessage>
<!--RAD3 Order Status Update from DSSOF to OP-->
<BeginTransaction tid="RAD3" cpaID="HL7_INT2"/>
<ReceiveMessage id="R03_a" transactionID="RAD3" fromActor="DSSOF" description="Receive RAD-3 Order Status Update" >
   <Transport transportHandler="HL7MLLPReceiver">
   </Transport>
   <VariableRef>hl7Message</VariableRef>
</ReceiveMessage>
<TestAssertion id="TA3_a" description = "Check according to message profile">
   <ValidateContent validationAdapter="hl7xmladapter"
      schemaLocation="testsuites/Dicom_IHEPIR/profiles/ORM-O01-Profile.xml"
      description="The control of the ACK message with the HL7 message profile.">
      <Expression>$hl7Message</Expression>
   </ValidateContent>
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check if Order Control Code SC">
      <Expression>$hl7Message</Expression>
      <Expression>//ORC/ORC.1/text() = 'SC'</Expression>
   </VerifyContent>
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check if Order Status is Completed">
      <Expression>$hl7Message</Expression>
      <Expression>//ORC/ORC.5/text() = 'CM'</Expression>
   </VerifyContent>
</TestAssertion>
<Assign id="ASGN_32_a" variableRef="messageControlIdentifer" append="no_append">
   <Expression>$hl7Message//MSH/MSH.10/text()</Expression>
</Assign>
<SendMessage id="S03_a" description="Send an ACK Message" transactionID="RAD3" toActor="DSSOF">
   <Transport transportHandler="HL7MLLPSender">
   </Transport>
   <SetMessage description="ADT_Message">
      <Content>
 <Reference isDynamicContent = "true" mime_type="text" href="testsuites/Dicom_IHEPIR/msgtemplates/RAD12/ADT-ACK.txt"/>
      </Content>
   </SetMessage>
</SendMessage>
<EndTransaction tid="RAD3"/>
<!-- Response to RAD7 Performed Procedure Step Completed from DSSOF to Modality-->
<ReceiveMessage id="R08" transactionID="RAD7" fromActor="DSSOF" description = "Please send the RAD 7 (NSET RSP MPPS)">
   <Transport transportHandler="DICOMReceiver">
      <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
      <ConfigurationAttribute name="presentationContextID">1.2.840.10008.3.1.2.3.3</ConfigurationAttribute>
      <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
      <ConfigurationAttribute name="releaseWhenFinished">true</ConfigurationAttribute>
   </Transport>
   <VariableRef>assoc_rq</VariableRef>
   <VariableRef>assoc_ac</VariableRef>
   <VariableRef>pdv_command</VariableRef>
   <VariableRef>pdv_dataset</VariableRef>
   <VariableRef>a_abort</VariableRef>
</ReceiveMessage>
<Assign id="ASGN_08" variableRef="cfind_dataset_xml" append="no_append">
   <Expression>$pdv_command</Expression>
</Assign>
<TestAssertion id="TA8" description = "Validate DICOM Content according to Table F.7.2-1">
   <VerifyContent verificationAdapter="dicomadapter"
      description="Validate DICOM Content">
      <Expression>$MPPSCUID</Expression>
      <Expression>$n-set-rsp</Expression>
      <Expression>$pdv_command</Expression>
      <Expression>$pdv_dataset</Expression>
   </VerifyContent>
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check whether Image Manager returns ack code properly.">
      <Expression>$cfind_dataset_xml</Expression>
      <Expression>/dicom/attr[@tag='00000900']/text() = '0'</Expression>
   </VerifyContent>
   <VerifyContent verificationAdapter="xpathadapter"
      description="Check whether Image Manager returns response message id properly.">
      <Expression>$cfind_dataset_xml</Expression>
      <Expression>/dicom/attr[@tag='00000120']/text() = '1387'</Expression>
   </VerifyContent>
</TestAssertion>
<EndTransaction tid="RAD7"/>
<!-- RAD6 Performed Procedure Step In Progress from Modality to DSSOF for Unscheduled Case-->
<BeginTransaction tid="RAD6-2" cpaID="DICOM_INT2"/>
<Assign id="ASGN_6a-2" variableRef="cfind_dataset_xml" append="no_append">
   <Expression>#testsuites/Dicom_IHEPIR/msgtemplates/SWF_OF_Modality_Interface/mpps-ncreate-unscheduled.xml</Expression>
```

```
  </Assign>
  <Assign id="ASGN_7a-2" variableRef="cfind_dataset_dcm" append="no_append">
     <Expression>$cfind_dataset_xml</Expression>
  </Assign>
  <SendMessage id="S12-2" description="TestBATN_SCP will send an N_CREATE_REQUEST to your application" transactionID="RAD6-2" toActor="DSSOF">
     <Transport transportHandler="DICOMSender">
        <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
        <ConfigurationAttribute name="presentationContextID">1.2.840.10008.3.1.2.3.3</ConfigurationAttribute>
        <ConfigurationAttribute name="remoteAETitle">DCM_SERVER</ConfigurationAttribute>
        <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
     </Transport>
     <Packaging packagingHandler="DIMSESender">
        <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
     </Packaging>
     <SetMessage description="N_CREATE REQUEST">
        <Content><VariableRef>n-create-rq</VariableRef></Content>
        <Content>
   <VariableRef>cfind_dataset_dcm</VariableRef>
        </Content>
        <Content><VariableRef>msgId</VariableRef></Content>
        <Content><VariableRef>MPPSCUID</VariableRef></Content>
        <Content><VariableRef>iuid2</VariableRef></Content>
     </SetMessage>
  </SendMessage>
  <ReceiveMessage id="R12-2" transactionID="RAD6-2" fromActor="DSSOF" description = "Please send the RAD 6 (NCREATE RSP MPPS)">
     <Transport transportHandler="DICOMReceiver">
        <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
        <ConfigurationAttribute name="presentationContextID">1.2.840.10008.3.1.2.3.3</ConfigurationAttribute>
        <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
        <ConfigurationAttribute name="releaseWhenFinished">true</ConfigurationAttribute>
     </Transport>
     <VariableRef>assoc_rq</VariableRef>
     <VariableRef>assoc_ac</VariableRef>
     <VariableRef>pdv_command</VariableRef>
     <VariableRef>pdv_dataset</VariableRef>
     <VariableRef>a_abort</VariableRef>
  </ReceiveMessage>
  <Assign id="ASGN_08a-2" variableRef="cfind_dataset_xml" append="no_append">
     <Expression>$pdv_command</Expression>
  </Assign>
  <TestAssertion id="TA12-2" description = "Validate DICOM Content according to Table F.7.2-1">
     <VerifyContent verificationAdapter="dicomadapter"
        description="Validate DICOM Content">
        <Expression>$MPPSCUID</Expression>
        <Expression>$n-create-rsp</Expression>
        <Expression>$pdv_command</Expression>
        <Expression>$pdv_dataset</Expression>
     </VerifyContent>
     <VerifyContent verificationAdapter="xpathadapter"
        description="Check whether Image Manager returns ack code properly.">
        <Expression>$cfind_dataset_xml</Expression>
        <Expression>/dicom/attr[@tag='00000900']/text() = '0'</Expression>
     </VerifyContent>
     <VerifyContent verificationAdapter="xpathadapter"
        description="Check whether Image Manager returns response message id properly.">
        <Expression>$cfind_dataset_xml</Expression>
        <Expression>/dicom/attr[@tag='00000120']/text() = '1387'</Expression>
     </VerifyContent>
  </TestAssertion>
  <EndTransaction tid="RAD6-2"/>
<!-- RAD7 Performed Procedure Step Completed from Modality to DSSOF for Unscheduled Case-->
  <BeginTransaction tid="RAD7-2" cpaID="DICOM_INT2"/>
  <Assign id="ASGN_6-2" variableRef="cfind_dataset_xml" append="no_append">
```

```
    <Expression>#testsuites/Dicom_IHEPIR/msgtemplates/SWF_OF_Modality_Interface/mpps-nset-unscheduled.xml</Expression>
 </Assign>
 <Assign id="ASGN_7-2" variableRef="cfind_dataset_dcm" append="no_append">
    <Expression>$cfind_dataset_xml</Expression>
 </Assign>
 <SendMessage id="S08-2" description="TestBATN_SCP will send an N_SET_REQUEST to your application" transactionID="RAD7-2" toActor="DSSOF">
    <Transport transportHandler="DICOMSender">
       <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
       <ConfigurationAttribute name="presentationContextID">1.2.840.10008.3.1.2.3.3</ConfigurationAttribute>
       <ConfigurationAttribute name="remoteAETitle">DCM_SERVER</ConfigurationAttribute>
       <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
    </Transport>
    <Packaging packagingHandler="DIMSESender">
       <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
    </Packaging>
    <SetMessage description="N_SET REQUEST">
       <Content><VariableRef>n-set-rq</VariableRef></Content>
       <Content>
  <VariableRef>cfind_dataset_dcm</VariableRef>
       </Content>
       <Content><VariableRef>msgId</VariableRef></Content>
       <Content><VariableRef>MPPSCUID</VariableRef></Content>
       <Content><VariableRef>iuid2</VariableRef></Content>
    </SetMessage>
 </SendMessage>
 <ReceiveMessage id="R08-2" transactionID="RAD7-2" fromActor="DSSOF" description = "Please send the RAD 7 (NSET RSP MPPS)">
    <Transport transportHandler="DICOMReceiver">
       <ConfigurationAttribute name="aeTitle">TestBATN_SCP</ConfigurationAttribute>
       <ConfigurationAttribute name="presentationContextID">1.2.840.10008.3.1.2.3.3</ConfigurationAttribute>
       <ConfigurationAttribute name="transferSyntaxID">1.2.840.10008.1.2</ConfigurationAttribute>
       <ConfigurationAttribute name="releaseWhenFinished">true</ConfigurationAttribute>
    </Transport>
    <VariableRef>assoc_rq</VariableRef>
    <VariableRef>assoc_ac</VariableRef>
    <VariableRef>pdv_command</VariableRef>
    <VariableRef>pdv_dataset</VariableRef>
    <VariableRef>a_abort</VariableRef>
 </ReceiveMessage>
 <Assign id="ASGN_08-2" variableRef="cfind_dataset_xml" append="no_append">
    <Expression>$pdv_command</Expression>
 </Assign>
 <TestAssertion id="TA8-2" description = "Validate DICOM Content according to Table F.7.2-1">
    <VerifyContent verificationAdapter="dicomadapter"
       description="Validate DICOM Content">
       <Expression>$MPPSCUID</Expression>
       <Expression>$n-set-rsp</Expression>
       <Expression>$pdv_command</Expression>
       <Expression>$pdv_dataset</Expression>
    </VerifyContent>
    <VerifyContent verificationAdapter="xpathadapter"
       description="Check whether Image Manager returns ack code properly.">
       <Expression>$cfind_dataset_xml</Expression>
       <Expression>/dicom/attr[@tag='00000900']/text() = '0'</Expression>
    </VerifyContent>
    <VerifyContent verificationAdapter="xpathadapter"
       description="Check whether Image Manager returns response message id properly.">
       <Expression>$cfind_dataset_xml</Expression>
       <Expression>/dicom/attr[@tag='00000120']/text() = '1387'</Expression>
    </VerifyContent>
 </TestAssertion>
<EndTransaction tid="RAD7-2"/>
   </Thread>
</ThreadGroup>
```

```
<ThreadGroupRef nameRef = "TG01"/>
</TestCase>
```

VITA

PERSONAL INFORMATION

Surname, Name:	NAMLI, Tuncay
Nationality:	Turkish (TC)
Date and Place of Birth:	10 November 1982, Artvin
Marital Status:	Single
Phone:	+90 312 210 1763
e-Mail:	tuncay@srdc.com.tr

EDUCATION

Degree	Institution	Year of Graduation
MS	METU-Computer Engineering	2007
BS	METU-Computer Engineering	2005
High School	Artvin Anatolian High School, Artvin	2000

WORK EXPERIENCE

Year	Place	Enrollment
2008-present	SRDC, Reseach, Development and Consultancy Ltd	Researcher
2005-2008	Software Research and Development Center, METU	Researcher
2004-2005	Software Research and Development Center, METU	Part Time Software Developer

FOREIGN LANGUAGES

Advanced English

PUBLICATIONS

1. Maohua Yang, Christian Lüpkes, Asuman Dogac, Mustafa Yuksel, Fulya Tunçer, Tuncay Namli, Manuela Plößnig, Jürgen Ulbts, Marco Eichelberg, “iCARDEA - an Approach to Reducing Human Workload in Cardiovascular Implantable Electronic Device Follow-Ups”, Computing in Cardiology 2010, Volume 37, Page 221- 224, Alan Murray, ISSN 0276-6574.

2. Namli, T., Dogac, A., “Testing Conformance and Interoperability of eHealth Applications”, Methods of Information in Medicine, Vol. 49, No.3, May 2010, pp. 281-289.

3. Bergengruen O., Fischer F., Namli T., Rings T., Schulz S., Serazio L., Vassiliou T., “Ensuring Interoperability with Automated Interoperability Testing”,White Paper, 2010, European Telecommunications Standards Institute.

4. Namli T., Aluc G., Dogac A., “An Interoperability Test Framework for HL7 based Systems”, IEEE Transactions on Information Technology in Biomedicine Vol.13, No.3, May 2009, pp. 389-399.

5. Kabak Y., Olduz M., Laleci G. B. Namli T., Bicer V., Radic N., Dogac A., “A Semantic Web Service Based Middleware for the Tourism Industry”, Book Chapter in Semantic Enterprise Application Integration for Business Processes: Service-Oriented Frameworks, Edited by G. Mentzas and A. Friesen, Business Science Reference (an imprint of IGI Global), USA, 2009, pp. 189-211.

6. Namli T., Dogac A., Sinaci A. A., Aluc G., “Testing the Interoperability and Conformance of UBL/NES based Applications”, eChallenges Conference, October 2009, Istanbul, Turkey.

7. Ivezic N., Cho H., Woo J., Shin J., Kim J., Abidi M., Dogac A., Namli T., Legner C., Fischer F. “Towards a Global Interoperability Test Bed for eBusiness Systems”, eChallenges Conference, October 2009, Istanbul, Turkey.

8. Namli T., Aluc G., Sinaci A., Kose I., Akpinar N., Gurel M., Arslan Y., Ozer H., Yurt N., Kirici S., Sabur E., Ozcam A., Dogac A., “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.

9. Dogac A., Kabak Y., Namli T., Okcan A., “Collaborative Business Process Support in eHealth: Integrating IHE Profiles through ebXML Business Process Specification Language”, IEEE Transactions on Information Technology in Biomedicine, Vol.12, No.6, November 2008, pp. 754-762.

10. Namli T., Dogac A., “Using SAML and XACML for Web Service Security and Privacy”, Book Chapter in Securing Web Services: Practical Usage of Standards and Specifications, Edited by P. Periorellis, Idea Group Publishing, USA, 2008, pp. 183-206.

11. Thiel A., Eichelberg M., Wein B., Namli T., Dogac A., “Goals and Challenges for the realization of a European wide eHealth infrastructure”, European Conference on eHealth (ECEH07), Oldenburg, Germany, October 2007

12. Dogac A., Namli T., Okcan A., Laleci G., Kabak Y., Eichelberg M., “Key Issues of Technical Interoperability Solutions in eHealth and the RIDE Project”, eChallenges Conference, The Hague, The Netherlands, October 2007

13. Della Valle E., Cerizza D., Celino I., Dogac A., Laleci G., Kabak Y., Okcan A., Gulderen O., Namli T., Bicer V. , “An eHealth Case Study”, Book Chapter in ”Semantic Web Services: Concepts, Technologies, and Applications”, Studer, Rudi; Grimm, Stephan; Abecker, Andreas (Eds.), 2007, Approx. 15 p., 100 illus., Hardcover, ISBN: 978-3-540-70893-3, Due: April 5, 2007, Springer.

14. Dogac A., Kabak Y., Gulderen O., Namli T., Okcan A., Kilic O., Gurcan Y., Orhan U., Laleci G., “ebBP Profile for Integrating Healthcare Enterprise (IHE)”, Submitted to OASIS ebXML Business Process Technical Committee.

15. Namli, T., Dogac A., “Implementation Experiences On IHE XUA and BPPC”, Technical Report, December 2006.

16. Laleci G., Dogac A., Akcay B., Olduz M., Yuksel M., Orhan U., Tasyurt I., Sen T., Kabak Y., Namli T., Gulderen O., Okcan A., “SAPHIRE: A semantic Web service based Clinical guideline deployment infrastructure exploiting IHE XDS”, eChallenges Conference, Barcelona, Spain, October 2006. Published in: Exploiting the Knowledge Economy: Issues, Applications, Case Studies, Paul Cunningham and Miriam Cunningham (Eds), IOS Press, 2006 Amsterdam, ISBN: 1-58603-682-3.