Testing Interoperability Standards – A Test Case Generation Methodology

Nenad Ivezic* — Jungyub Woo*

**100 Bureau Drive*
Gaithersburg, MD 20899
United States

nenad.ivezic@nist.gov
jungyub.woo@nist.gov

ABSTRACT: *Over many years, National Institute of Standards and Technology (NIST) built test beds to support interoperability standards development and their implementation within software applications. A general test framework has been proposed to enhance new test bed development and reuse of existing test components and materials. Currently, the test framework is undergoing a validation effort within a healthcare domain to develop a test case generation facility.*

KEY WORDS: *interoperability, standards, testing, test bed, test framework, NIST.*

1. Introduction

National Institute of Standards and Technology (NIST) built numerous test beds to support interoperability standards development and their implementation within software applications for industries such as automotive, construction, and healthcare.

Currently, individual test beds are being built almost from scratch with very limited reuse of existing test materials and components. This is not cost effective and is problematic from the perspective of advancing knowledge of testing. Existing testing approaches and frameworks do not address the issue of reuse to any significant extent (IIC, 2010; RosettaNet 2004; TTCN-3, 2010; TaMIE, 2010).

For that reason, NIST started to develop a general test framework to provide a vehicle for generalization and accumulation of knowledge of testing while providing a platform for management and reuse of test materials and test components.

Presently, we are validating the test framework on a number of industrial test cases. In this paper, we describe an application of the test framework to establish a unified test case generation methodology and to design an architectural solution for a supporting test case generation tool.

2. A General Test Framework: Test Case Design

A key consideration when developing a test framework is that it should support a variety of alternative testing modes including independent document validation, individual application conformance, and peer-to-peer (or interoperability) testing. Also, the test framework should allow easy adaptation of test materials to any one of the above testing scenarios.

The NIST-proposed test framework allows capture of test materials from the underlying domain and business perspectives and without reference to a specific testing configuration or role that test components and system under test (SUT) may assume (Ivezic *et al.*, 2010). This is in contrast to most existing test case designs that depend on both a standard specification and a specific test bed implementation or testing configuration.

Additionally, to allow easy adaptation of the test materials, the test framework includes a test case architecture containing two layers: an abstract test case and executable test case. An abstract test case is derived, in general, from standard specifications and the intended usage patterns for the system under test. Its purpose is to specify the validation rules and testing procedure at an abstract level. Validation rules are written using logical conditions; that is, they describe the normative requirements based on the standard specifications. The testing procedure describes the usage patterns that are simulated for the SUT.

Abstract test cases are intended for human consumption and may be thought of as a meta-model for the executable test cases. This implies that the abstract test case is independent of a specific test bed implementation and testing configuration. On the other hand, an executable test case is an implementation of the abstract test case that executes the validation process. Consequently, the executable test case contains machine-readable content that reflects a specific test bed and test configuration.

Another key issue for existing test frameworks is that a typical test case design embeds verification rules as an integral part of the testing procedure. These verification rules are used to ascertain whether the test items are true with respect to the test requirements. In this way, these two parts of the test case are closely coupled, because the verification rules will be executed at a specific point in time within the testing procedure. This approach, however, gives rise to two types of problems. First, test cases tend to be monolithic, large, and difficult to maintain. Also, test case design is difficult to modify when the underlying standards change. The second problem is low reusability. Since verification rules are based on the SUT test requirements and testing is based on the business scenarios in which the SUT participates, numerous combinations are possible. The tight coupling means that each such combination will require significant changes to the test cases.

To overcome these problems the NIST-proposed test framework contains a modular design for test cases, in which the test cases consist of procedural contents and verification rules. Each test case (either the abstract or executable test case) is

composed of two scripts. One script contains procedural content: a usage script for the abstract test case and a procedure script for the executable test case. The other script contains verification content: an assertion for the abstract test case and a verification script for the executable test case.

The two procedural scripts are distinguished by their intent and time of specification. The usage script in the abstract test case represents the testing-related business process, which includes the partners' life cycles and actions during testing. Actions are abstract descriptions and contain no message instances. For example, the usage script may say "Buyer sends a purchase order message to a Supplier." The specific buyer, purchase order, and supplier instances are not yet specified. On the other hand, the procedural script in the executable test case represents a business transaction that will be executed and contains specific instances and references to the actors in the business process.

Verification scripts contain event-driven conditions, which must be satisfied before the verification script is activated (triggered). When the activation condition is satisfied, a test item, such as a document or a message, is verified against an assertion. These activation conditions render the verification rule independent of the testing procedure, since the rule is not activated at a specific step of testing procedure. Consequently, verification scripts may be reused readily within a new testing procedure because the verification script is independently executed by the events during the test procedure.

Verification scripts are distinguished by their intent and time of specification. The assertion script in the abstract test case is human-, not machine-, readable because a specific verifier may be unknown at development time. When that verifier is known, the Test Case Developer can add assertion codes using an executable language. This assertion code is the verification script in the executable test case.

3. A Test Framework Validation: Test Case Generation Tooling

Test generation, within any realistically complex domain, is a complex task that involves management of evolving testing requirements, capturing correct intent of these requirements, and efficient management of change in any aspect of the testing process. Within the healthcare domain, NIST is developing a test case generation methodology that can span numerous healthcare sub-domains and interoperability profiles. The NIST framework is used to architect the test case generation tooling.

The essential goal for a test case generation tool is to facilitate specification, generation, and traceability of test cases. Specification entails representation of testing requirements in an abstract form that enables computational assessment whether a system under test has met the requirements. Generation entails transformation of an abstract test form into an executable form that may be run on a specific computational platform. Traceability entails capturing relationships among

requirements, decisions made in the testing execution environment, the resulting abstract test forms, and executable forms of test cases. Additional usability and operational requirements for the facility include maintenance of complete specifications in a so called intermediate form. This form maintains complete information required to create executable test cases.

Figure 1 illustrates the workflow that the facility will support and its three operational stages. At Stage 1, the facility enables interactions with users in support of test case specification. Here, test case requirements are captured through interaction with the Test User to obtain Test Case Setup information and with the Test Case Developer to identify key Test Events from the underlying Test Requirements materials. As a result of this stage, the Abstract Test Case Repository is populated for the testing objectives at hand. In addition to the previously introduced Usage Script and Assertion Script test artefacts, the Message Template artefact is introduced with the role to provide schema-type constraint information for the test messages to be created. Since the Abstract Test Case does not consider test specifics, the message template has no specific values assigned and will be used to generate a message instance or validation context file at Stage 2.

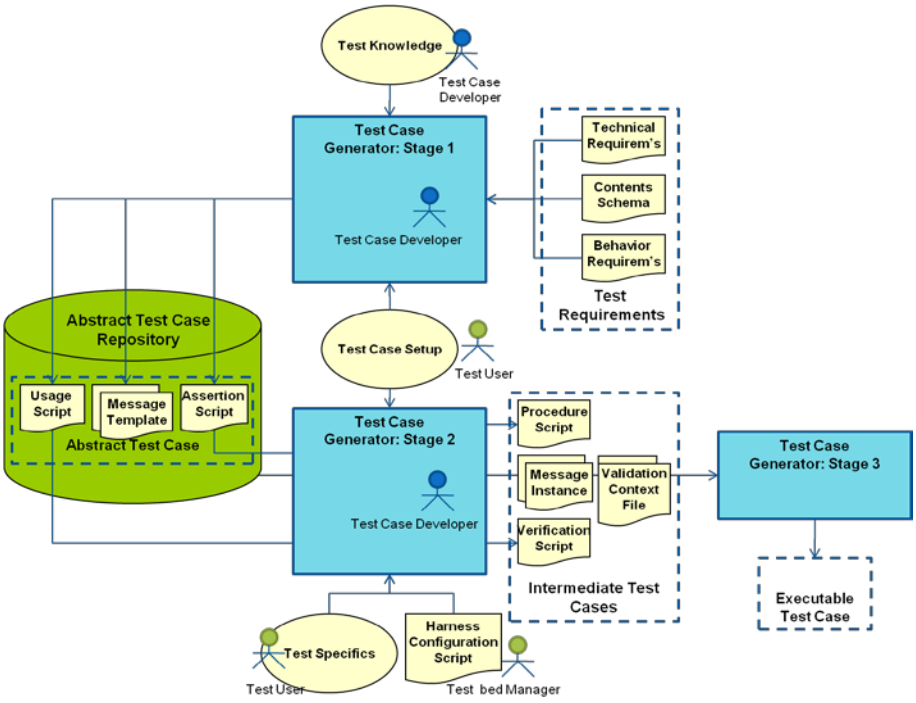

Figure 1. *Test Case Generator Facility and Its Operational Stages*

At Stage 2, the facility enables interactions with users in support of additional specification of the test execution environment. The run-time information is captured through interaction with the Test User's additional setup information as well as the Test Specifics, e.g., test model or test message data definition. The test harness configuration and role assignments of the test modules, such as validation service or message handling, and SUT(s) are specified through interaction with the Test Bed Manager. The Procedure Scripts identify required message instances that will be sent during testing. Additionally, the facility provides a graphical user interface for Test Users to generate message instances from the message templates. To validate a message from a SUT, the Test Case Developer uses the message template to create a validation context file with expected values for the test object sent by the SUT. The outcome of this stage is the Intermediate Test Case collection with sufficient information about test cases and execution environment to support automated generation of Executable Test Cases at Stage 3.

To be successful, the framework and the test generation facility will have to be supportive of a wide range of testing use cases across the different industries. An iterative prototype-evaluate-refine approach to the validation of test generation facility is adopted with an initial focus on the healthcare industry.

4. Impacting Testing Interoperability Standards

When testing interoperability standards, an organization utilizes different modes of testing including independent document validation, individual application conformance, and peer-to-peer/interoperability testing. The exact testing strategy for interoperability standards will depend on many factors such as the complexity of applications, size of the community, the time horizon for implementing, management of the standards, and so on.

Traditionally, the underlying test materials were captured within procedural statements tied to a specific testing mode as well as a specific testing configuration. This approach has proven to be unwieldy and hard to manage because the logical definitions of correctness of an implementation are buried within the testing procedures that deal with run-time testing issues such as test bed configuration and test material execution.

With the proposed test framework facility and the test case design in place, the logic of testing is not encumbered by the specific testing mode, configuration, and execution concerns any more. As a consequence, it is much easier to move from one mode of testing to another, from one test bed configuration to another (e.g., from a single simulation node to multi-simulation node test bed), and from one type of execution environment to another (e.g., from one configuration of Web services to another).

From the perspective of advancing the knowledge of testing interoperability standards, the proposed testing framework and test case design are initial steps in the direction of abstracting and organizing the testing knowledge for greater efficiency and transparency. Without a concentrated effort to agree on conceptualizations in the testing space, the ability to share and reuse test components and test beds for interoperability standards testing will continue to be very limited.

5. Conclusion

Presently, test case designs give rise to closely coupled, monolithic, and difficult-to-maintain test cases. Since test cases are difficult to modify when the underlying standards change, low reusability of test materials follows. To overcome these problems, the proposed test framework contains a modular test case design where the test cases consist of procedural content and verification rules. Also, the framework introduces the notion of abstract test case, which is independent of a specific test bed implementation. In the first validation of the test framework, the test framework methodology is assessed in the context of health care scenarios and the requirements to support specification, generation, and traceability of test cases.

Acknowledgements

Certain commercial software products identified in this paper were used only for demonstration purposes. This use does not imply approval or endorsement by NIST, nor does it imply these products are necessarily the best available for the purpose.

6. Bibliography

Ivezic, N., Woo, J., Cho, H., "Towards Test Framework for Efficient and Reusable Global e-Business Test Beds", In Proceedings of I-ESA 2010 Conference, Coventry, UK, 2010.

IIC ("ebXML IIC Test Framework Version 1.0." OASIS), on line at http://www.oasis-open.org/committees/download.php/1990/ebXML-TestFramework-10.zip, accessed March 2010.

RosettaNet (RosettaNet Ready Self-Test Kit (STK) User's Guide Release Version 2.0.7). RosettaNet, 2004.

TTCN-3 site, on line at http://www.ttcn-3.org, accessed March 2010.

TaMIE Site, on line at http://www.oasis-open.org/committees/tamie, accessed March 2010.