Testing and Monitoring E-Business using the Event-driven Test Scripting Language

J. Durand[1], S. Kulvatunyou[2], J. Woo[2], and Monica, J. Martin[3]

[1] Fujitsu Computer Systems, Enterprise Software & Solutions Group, 1250, E. Arques Ave., Sunnyvale, CA 94085, USA
jdurand@us.fujitsu.com

[2] Manufacturing Systems Integration Division, National Institute of Standards and Technology, 100 Bureau Dr. Gaithersburg, MD 20899-8263, USA
serm@nist.gov, jwoo@nist.gov

[3] Sun Microsystems, 4140 Network Circle, Santa Clara, CA 95054, USA 95054
monica.martin@sun.com

Abstract. This paper addresses challenges associated with conformance and interoperability testing of today's e-business technologies and proposes a new approach that improves on existing, test scripting languages and operation modes. A test model and scripting are described, namely, the Event-driven Test Scripting Language (eTSL). We contrasted this new approach with existing testing technologies and provide examples that illustrate the use of both the model and the language.

1. Background

The e-business/e-government technology space has been a very active, but fragmented area in terms of protocols and standards. Several messaging protocols are in use today and there has been a recent surge of Internet-based and SOAP-based standards. The collaboration aspects of partner interactions appear to be converging toward well-established types of message-exchange patterns such as those described in [2]. No unique executable representation prevails, however, besides generic modeling such as the Unified Modeling Methodology [3].

At a level more related to application domains, many industry verticals and government agencies have made attempts to facilitate semantic interoperability by (a) standardizing their business documents, taxonomies and collaboration patterns

(usually in XML), and (b) adopting messaging infrastructures that are based on open, rather than proprietary, standards. The result has been the emergence of many different suites of standards that claim to support interoperability.

Such a heterogeneous standards environment implies that conventional testing approaches are inadequate. Either they propose a general but unsupportive scripting platform (e.g., JXUnit [4]) that requires much effort in developing individual test cases for e-business, or they focus on a single aspect of interoperability only. For example, some test frameworks and languages are focusing on the communication protocol (e.g., TTCN-3 [6]), others on a specific messaging protocol, while ignoring the business aspect and the way this aspect may impact the use of the protocol and messaging features.

2. Requirements for Advancement

2.1 Testing of Integrated Standards through Deployment Profiles

E-business partners must agree on many different standards to interoperate. Typically, these standards are grouped into three layers.

- Messaging infrastructure standards, ranging from transport level - e.g., HTTP [21], SMTP [22] - to higher-level messaging protocols and quality of service (QoS) including reliability and security, such as those defined as SOAP extensions.
- Multi-message exchanges as manifested in business processes and choreographies. These include those which conform to UMM [3] business transaction patterns, ebXML Business Process Specification Schema (BPSS or ebBP) [7], WS-Choreography [8] or WS-BPEL [9]. .
- Business document standards. These are related industry-specific or horizontal business message standards. These can be the message model itself, the taxonomies in use, code lists, or the XML schema modeling style).

Deleted: [21]

Deleted: [22]

There have been conformance and interoperability testing tools and initiatives for each group individually, but approaches for testing *integration* across all groups has been ad-hoc at best. Such an approach requires more than just testing each layer separately. Indeed, it also means more than just using them together. It means *profiling* them in a way that is specific to application domain. Such profiles are called here *standards integration and deployment profiles* (or SID-profiles).

An SID-profile determines the options and features used in each selected standard, and adds user-defined conventions. In the area of Web services protocols, WS-I [17] has also defined profiles for integrating standards, but these remain orthogonal to user preferences and business concerns. A SID-profile may build upon a WS-I profile, but it has a broader reach: its user-defined conventions may involve business content, as well as rules on how standards' features and business conventions must be used together. For example, a user community may decide

that a higher level of message reliability and security must be used for all business transactions that involve money transfers.

An SID-profile also takes into account user customization and usage conventions decided at deployment time. In this regard it is close to the concept of deployment profile as defined in OASIS IIC [23] for the ebXML standard. From a contractual perspective, the SID-profile represents the technical aspects of an interoperability agreement between parties. These technical aspects impact all of the aforementioned layers, leading to a requirement that test cases be integrated across these layers.

2.2 Versatile Testing and Open Test Framework

The test case model and scripting defined in this document must be usable for two different purposes:

- Conformance and interoperabiltiy tests of e-business endpoints according to the SID-profile adopted by a user community. The objective is to prequalify endpoints before deployments. In this context, the script may require some controls over the e-business endpoints (e.g., ability to command an endpoint to initiate messages).
- Continuous monitoring of in-production e-business systems. The objective is to verify the normal operation of actual business transactions according to their specifications or to troubleshoot interoperability problems. In this case, the test script will not interfere or control the e-business endpoints. Test cases must allow for the validation to be executed *live* – at the same time the busines transaction unfolds – or *deferred* to a later time.

In both cases, the test script and event model must accommodate various messaging protocols. In addition, the test framework must be extensible with new protocol-specific or document-specific processing modules.

3. The eTSL Model for Testing and Monitoring

3.1 Architecture and Concepts

The eTSL execution model is based on the concepts of event and workflow. The main notions are described below, some of which are inhereted from the IIC Test Framework 1.1 [10]:

- **Monitor.** A monitor is a single-threaded workflow script controlling the execution of test steps, with event capabilities (listening and generating). A test case is represented by one or more monitors that may execute and synchronize concurrently.
- **Trigger**. A trigger is a control of a test case execution that defines under which conditions and events a monitor execution takes place. A trigger reacts either to events in the event board (event watching) or to a scheduled

date/time (clock watching). The general execution flow and timing of several test cases is then controlled by triggers, the set of which represents a test suite.

- **Event Board**. Event management, central to eTSL, is done by an event board, which supports event logging, search and correlation. The event board normally suffices for mediating all test-case inputs and outputs.
- **Event-Adapters**. E-business message traffic is mapped to and from events. Event-adapters perform these mappings, allowing for abstraction of test cases from specific communication protocols.
- **Evaluation-Adapters**. eTSL is designed to delegate some test result evaluations to external modules because different protocols need different evaluation engines. For example, evaluating a messaging protocol typically requires a simple pattern matching. Conversely, evaluating a business document or business process typically requires a rule-based or reasoning engine. Evaluation adapters manage this delegation process.

There are four underlying design principles for the eTSL. It is XML-based, simple, event-centric, and protocol agnostic.

- **XML-based**. The test script and expression rely on XML syntaxes and related technologies such as XPath [11] and XQuery [12]. This allows the scripting to work naturally with most of today's specifications, which are XML-based. Even if a message protocol is not XML-based, an event-adapter can wrap it into an XML format.
- **Simplicity**. The goal is to make eTSL implementations easy to validate. For example, a minimal set of control features sufficient for e-business testing has been selected, rather than replicating the full range of conventional workflow operators.
- **Event-centric.** All traffic must be captured in the form of events and wrapped into a standard XML event envelope - as opposed to just business messages as in the IIC test framework. This allows us to use the same test script in either a deferred or live-validation context. The coordination of test-case executions within a test suite is also event-driven; e.g., a trigger may start a new test case by watching for the final-status event of the previous test case. The state of the test case workflow (e.g. variables, last step executed) is also represented as events so that no additional persistence mechanism is required by a recoverable test engine.
- **E-business protocol agnostic**: Test script logic and control are abstracted from e-business protocols; it is versatile for messaging, business process, and business content testing regardless of technologies. Hence it can be used with either ebXML [13] or Web Services message profiles [14]. Of course a test case script that verifies business headers in ebXML may not apply to Web service messages, but often a change in event-adapter is the only modification needed to adapt a test script focused on, say, verifying business transaction and payloads, from one message protocol to the other.

The general architecture and execution model for eTSL script execution is illustrated in Figure 1. The test case begins with a trigger, which is designed to

Deleted: Figure 1

listen to a particular event such as a time point or a message. This initiates the root monitor, which oversees test-step execution by looking for interested events on the event board. The event board gets events from both the monitor (internally generated) and the event adapter (intercepted communication traffic). The monitor may start child monitors synchronously or asynchronously to look for related events. After all child and root monitors exit, the test result is generated.

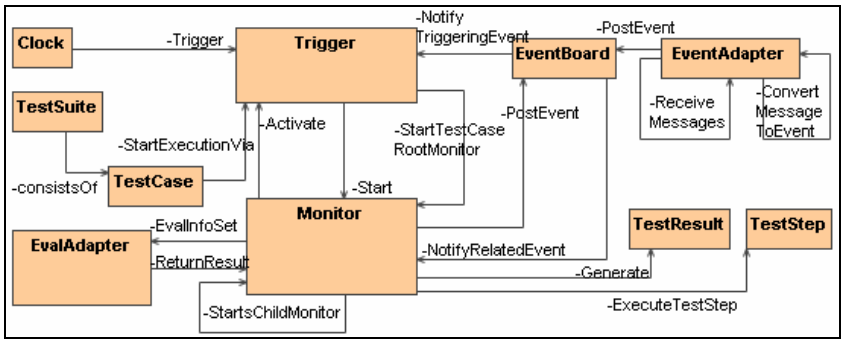

Figure 1: General architecture and execution model of the eTSL

3.2 Scripting Primitives

Triggers and monitors are scripted in XML. Each of the test steps executed by a monitor workflow as a workflow activity, is an atomic unit of work. Eleven types of workflow activities are defined in [15] based on the experience described in [10]. They are grouped into four classes: event operations, monitor flow control, external resources, and test case control, which are described briefly below.

Event operations:

- **post:** generate an event from testing or monitoring material and post to the Event Board.
- **find:** select one or more non-masked 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.
- **mask:** mask or unmask some past events to a monitor instance.

Monitor flow control:

- **start:** starts a new instance of a monitor, either synchronously to the current instance or asynchronously.
- **set:** assigns a value or an XML infoset to a variable or to a constant. Some reserved variables are automatically assigned at the completion of most test steps (e.g. $status, $output) or at the end of a test case (e.g., $result). Some of these are used in the example shown in section 3.3.

- **sleep:** the monitor execution is suspended either until a specified date or for a specified duration. The duration may be virtual in case of deferred execution (see section 3.3).
- **cad:** check-and-do operation that verifies a predicate on the result of one or more previous test steps and does a single operation in case the verification is positive; e.g., exit the test case. It allows for optional branching to another labeled step after the "do" part.
- **jump:** pursue the execution thread at another (labeled) test step in the monitor.

External resources:

- **call:** either an event-adapter or an evaluation-adapter is invoked. The invocation of an event-adapter will result in a message generation/sending from an event or from input argument(s). Invoking an evaluation-adapter will produce an XML infoset.

Test case control:

- **actr:** dynamically activate a trigger.
- **exit:** terminates the current test case along with all associated monitor instances, with a result of either pass, fail or undetermined. A trace of the test case execution and status are posted as events to the event board.

3.3 Examples Test Scripts

Consider the following business transaction:

- Step 1: Buyer sends a purchase order (PO) to Supplier.
- Step 2: Buyer receives a related PO Confirmation or Rejection message.

Constraints: (c1) The total time for both steps must be less than 600 seconds. (c2) The message correlation is based on a PO number that is in the payload.
Validation conditions: (v1) within 600 seconds, exactly one message of either PO Confirmation or Rejection is received that properly correlates with the PO, and no failure occurred.
Failure cases: (f1) an error message is received, that correlates with the initial message based on messageId or (f2) more than 1 response message (referencing the same PO) received within 600 seconds, or (f3) no response received within 600 seconds.

A test case will be used to verify every business transaction of this type including the validation conditions and constraints, and absence of any failure case. The following test case is processing message materials related to one business exchange. It gets these materials in the form of events from the Event Board. It can be validated in real-time (live) or deferred. Note the explicit mention of which evaluation engine is required for elements in <selector> and <condition>. Here, XPath paths and logical expressions are indicated with <xpath>, <xplog> tags.

Additional engines may be plugged in and referred to with specific tags in the script.

```
<monitor label="mBuyerSupplier">
<!-- === step 1: get the first PO message in the Event Board, within the specified time
window -->
<find step="step1"><selector get="[position()=1]" starting=”initialDate”>
<xpath>event/content/soap/Header/msgData/[action="PO"]</xpath>
</selector></find>
<!-- === : set some variables to elements of found PO message, for later use -->
<set var="POref">$output/event/content/soap/body/PO/POReference</set>
<set var="mId">$output/event/content/soap/header/msgData/MsgId</set>
<!-- === : sleep 600 seconds -->
<sleep duration="600"/>
<!-- === step 2: get related error message(s) if any -->
<find step="step2"><selector starting=”initialDate”>
<xpath>event/content/soap/Header/msgInfo[type="Error"][RefToMsgId=$mId]</xpath>
</selector></find>
<!-- === step 3: check exit failure case F1 (one or more error messages) -->
<cad step="step3">
<condition><xplog>count($output/event) &gt; 0</xplog></condition>
<do><exit value="fail"/></do>
</cad>
<!-- === step 4: get related response message(s) if any -->
<find step="step4"><selector starting=”initialDate”>
<xpath>event/content/soap/Header/msgData[POReference=$POref] [action = "confirm"
or action = "reject"]</xpath>
</selector></find>
<!-- === step 5: check success case (exactly 1 response) -->
<cad step="step5">
<condition><xplog>count($output/event) = 1</xplog></condition>
<do><exit value="pass"/></do>
</cad>
<!-- === step 6: check exit failure case F2 (too many responses) -->
<cad step="step6">
<condition><xplog >count($output/event) &gt; 1</xplog ></condition>
<do><exit value="fail"/></do>
</cad>
<!-- === step 7: exit failure case F3 (no responses) -->
<exit step="step7" value="fail"/>
</monitor>
```

The above example shows that the validation conditions checks take a significant part of the script (steps 5, 6, 7). A more concise and declarative notation is considered in a next version, that would simplify the script and improve readability.

The test case execution will be controlled by the following trigger. This trigger is itself activated (operation <actr>) at a specific date/time (activation date) within a monitor named "init" that can be started via a command line interface. The activation date in the <context> element defines the earliest date at which events may be selected by the trigger in order to start a test case. The <repeat> element

will cause the trigger to iterate on the next PO event selected, and so on until the last PO of July 2006. At each iteration a new instance of the mBuyerSupplier monitor is started, which has an initialDate value set by the time of the triggering event. This demonstrates the continuous monitoring functionality of the eTSL.

```
<monitor label="init"><!-- === Next, just activate the trigger -->
<actr>
<context><activationDate>2006-07-01T00:00:01Z</activationDate></context>
<trigger name="JulyBatch" type="Event">
 <find step="step1"><selector get="[position()=1]">
<xpath>event/content/soap/Header/msgData/[action="PO"]</xpath>
 </selector></find>
 <repeat until="2006-07-31T23:59:59Z"/>
 <monitors><start casevar="mycesid" mlabel="mBuyerSupplier"/></monitors>
</trigger></actr></monitor>
```

The above test case can be run either "live" for validating business transactions while they are unfolding, or "deferred" for analyzing logs of past business transactions, without any change. Indeed, when all inputs and outputs of a monitor use the event board, there is no essential difference between the two modes of validation.

Case of deferred validation: The above test case can be run for analyzing past events; e.g., after the SUT has been shut down or after a period of run time. This would be the case if the "init" monitor is executed in August 2006. The activation date (July 2006) for the trigger "JulyBatch" would be past, and an instance of the mBuyerSupplier monitor will be started for every PO message logged in the EventBoard between July 1st and July 31st.

Case of live validation: If the "init" monitor is started prior to July 2006, then the trigger will activate automatically on July 1st, which will be the present date. An instance of the mBuyerSupplier monitor will start every time a new PO message is posted to the event board after July 1st, and the monitor will execute in sync with the transaction.

As a consequence of using the same execution model for live or deferred validation, a "live" test case may also execute partly in the deferred mode; e.g., after a recovery of the test engine that introduced delays in the monitoring. All executions are based on a virtual present time called *time-cursor* which may be set to a past date. In this case, the execution will "hop" from one past event to another (e.g., the <sleep> step in the monitor will not actually block the test run for 600 seconds) until the time-cursor reaches the present time. For each test case execution, a trace-event reporting the result (fail/pass/undetermined) and other details will be posted to the Event Board.

4. Related Works

This section analyzes existing works relating to test scripting in the realm of conformance and interoperability testing of communication systems, particularly

B2B systems. Because these types of tests target independent systems, they are done via a black box test. The related works we have identified are the ebXML IIC test framework [10], the RosettaNet self-test kit [16], the Web Services Interoperability (WS-I) test tools [17], the third version of the Testing Test Control Notation (TTCN-3) [6], Automatic Test Mark-up Language (ATML) [18] and the JXUnit (JUnit XML) test framework [4]. The rest of this section provide the analyses.

The OASIS ebXML **IIC test framework (TF v1.1)** was designed specifically to test the conformance and interoperability of the software implementing the ebXML Messaging Specification. The framework defines components and harnesses (configurations) necessary to simulate and control the environment for conformance and interoperability tests. The test is driven by a sequence of steps that contains specific commands interpreted by the simulated component to trigger the system under test (SUT) to perform necessary actions. The strengths of the IIC framework are its simplicity and the fact that it is one of the first test frameworks providing XML-native test scripting.

A shortcoming of the IIC TF is that it has been designed for ebXML messaging 2.0, and provides weak extensibility options, both for external events and for specialized evaluations. The IIC TF does not support other suites of B2B integration standards such as other SOAP profiles integrating several Web Services standards. It also does not support ebMS 3.0, which departs significantly from ebMS 2.0.

The **RosettaNet Self-Test Kit (RNSTK)** is a test system provided by the RosettaNet consortium. The system allows software solutions to perform self-tests for compliance to business collaboration scenarios and the RosettaNet Integration Framework (RNIF) specification [19]. The RNSTK tests an integrated system implementing particular B2B collaboration scenarios, yet is tightly dependent on RNIF messaging. RNSTK test suites are encoded using the RNIF specification itself. Due to these reasons, the RNSTK cannot be used to tests other suites of B2B standards including the Document Type Definition semantics [20]. Another weakness is that its architecture only supports conformance test configurations.

Deleted: [19]

Deleted: [20]

The **Web Services Interoperability (WS-I)** consortium was formed to address interoperability issues in Web Services specifications, particularly those issues arising from the integration of several such specifications. Toward that goal the consortium defines integration profiles and related test tools [17]. The unique characteristic these tools is that they do not follow or implement a test suite; instead, they apply a battery of predefined tests to Web service material provided as input. The objective is to verify the conformance of this material to one of the WS-I profiles. The tools only passively monitor message traffic and have no ability to control SUTs. Nevertheless, this monitoring architecture is noteworthy for its un-intrusive, simple, and deferred-analysis test environment. The analyzer tool of WS-I cannot be used as a general test engine, being tied to profile definitions; but, it can be leveraged by eTSL as an evaluation-adapter for verifying the conformance of message material to WS-I profiles.

The **TTCN-3** is a powerful, programmatically complete procedural language with specialized constructs for defining test procedures, test verdicts, matching mechanisms for evaluation, timer handling and distributed test components. Due to

its original focus on telecommunication systems, it also has the ability to specify encoding information, to communicate both synchronously and asynchronously, and to perform passive monitoring. These capabilities are all essential for testing in B2B integration. Relative to previous test frameworks, TTCN-3 is more powerful and flexible [1]. However, TTCN-3 notation can be difficult for a business or domain expert to use. TTCN-3 has not been designed to delegate some functions to external modules and it is weak when it comes to validating business transaction events and XML documents, which are essential for B2B integration testing.

ATML has been designed for exchange of diagnostic information, configuration data, and test instrument definitions between conforming software applications. Although it was not designed to support test scripting and execution, ATML representations could be complementary to test suites expressed in eTSL.

JXUnit is a general XML-based, test-scripting system built on top of the JUnit [5] Java classes. It is a data-driven testing system in which input to the SUT and expected output are specified and then the actual output is evaluated. There is no built-in support for B2B testing, in particular the communication and the event-driven tests. However, as a general, test-scripting platform that relies on a common programming language, JXUnit and JUnit could be used as an implementation platform for essentially any test framework including the one proposed here.

Table 1: Feature matrix of related test frameworks

Characteristics	eTSL	IIC 1.1	RNSTK	WS-I	TTCN-3	ATML	JXUnit
SUT control	Y	Y	N	N	Y	P	N
XML-oriented	Y	Y	Y	Y	N	N	P
Openness	P	N	P	N	Y	P	N
Protocol adaptability	Y	P	N	P	Y	Y	Y
SID targets	Y	P	N	P	Y	NA	N
Run-time versatility	L+D	L	L	D	L	NA	D
Event awareness	Y	P	N	N	Y	NA	N
Test Type	C+I	C+I	C	C+I	C+I	NA	C

Nomenclature: Y = Yes, N = No, P = Partial support, I=Interoperability, C = Conformance, L = Live testing, D = Deferred testing, NA = Not Applicable.

Table 1 summarizes the comparison between the eTSL and the aforementioned related works. The first column lists characteristics that are essential to address the challenges of e-business systems testing (See sections 1 and 2). These characteristics are described below. Although the table shows that eTSL supports all of these while other frameworks have partial support only, some caution should be used when interpreting this table:

Deleted: Table 1

(1) These characteristics should be read as design and functional objectives for eTSL that may not have been in the requirements of other test frameworks given their original purpose. The table shows where the approach used in these frameworks is inadequate for eTSL objectives.

(2) eTSL as a test execution model and scripting, needs be complemented with appropriate components in order to fulfill some of these characteristics (e.g. event-adapters for protocol adaptability)

SUT control: This characteristic indicates whether the test script has the ability to control the processing of the SUT - e.g. by simulating or altering input from external sources – in an automated way as part of the test case execution. The ATML is qualified as partial because it has the SUT control semantics but only for human consumption. The lack of this characteristic implies limited test automation and test coverage, particularly with the negative scenario.

XML-oriented: Native support for XML content in the test material. XML-oriented test frameworks have helpful functions to extract, analyze, and specify XML data. The support in JXUnit is partial because the script is in XML but there is no built-in support for XML test-data manipulation.

Openness: The test script's ability to provide controls to delegate processing to external modules and to define test-component configuration, which is also called testing mode. In the table, eTSL only provides delegation, while RNSTK provides only component configuration. WS-I test tools will be more open in their next version.

Protocol adaptability: The protocol-agnostic characteristic of the test script and the overall test framework. The support in IIC 1.1 is partial because it has built-in structures to cover a few existing standards but no semantics for new ones.

SID targets: This characteristic indicates the ability for the test script and framework to test SID-profiles - the integration of standards and user-defined conventions - without being tied to any one in particular. The support is only partial in IIC 1.1 because, even though it supports user conventions, it is not sufficiently open to standards other than ebXML. WS-I profiles may be seen as subsets of SID-profiles, hence the partial support.

Run-time versatility: This characteristic refers to the ability to run test cases at a different time than the actual e-business exchange run by the SUT (deferred validation vs. live).

Event awareness: The ability to drive and synchronize test execution and components with events and time conditions as well as ability to handle various artifacts, such as business messages, as events.

Test Type: Supported test cases may concern conformance of an e-business endpoint (to standard(s) and/or to user conventions) or the level of interoperability between e-business partners based on predefined exchange / transaction models. Conformance testing assumes the ability to simulate an endpoint behavior, while interoperability assumes the ability to capture and process the trace of a live exchange between partners.

5. Summary and Future Work

This paper has identified challenges associated with testing conformance and interoperability of e-business applications and proposed the concept of a Standards Integration and Deployment (SID) profile as target of this testing. It also proposed the eTSL approach as a pragmatic way to test for conformance and interoperability based on SID-profiles. The eTSL provides a concise, extensible, and readable test-case scripting, while keeping its set of constructs minimal and high-level. The most remarkable aspect of the eTSL is its use of events as a central concept rather

than just an auxiliary control feature. This versatility, combined with an open and extensible design, allows for the integrated testing required by e-business. The authors intend to propose the eTSL as an OASIS standard and to promote its adoption in leading B2B user-communities.

6. Disclaimer

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

7. References

[1] Schieferdecker, I. and Stepien, B., Automated Testing of XML/SOAP based Web Services. In Proceedings of the 13th. Fachkonferenz der Gesellschaft fur Informatik, Feb. 2003, 26-28.
[2] Hohpe, G. and Woolf, B.. "Enterprise Integration Patterns" Addison-Wesley signature series, Aug 2006.
[3] UNECE, UN/CEFACT Modeling Methodology R12, 2003.
[4] Java XML Unit (JXUnit), http://jxunit.sourceforge.net.
[5] JUnit, Java for Unit Test, http://junit.sourceforge.net.
[6] ETSI, Testing and Test Control Notation 3 Standard Suite, 2003.
[7] OASIS, Business Process Specification Schema 1.0.1, May 2001 and ebBP, v2.0.4, October 2006.
[8] Web Services Choreography Description Language (WSCDL), Version 1.0, (candidate recommendation) November 2005.
[9] OASIS, Web Services Business Process Execution Language 2.0 (committee draft in review phase), August, 2006.
[10] OASIS, ebXML Implementation, Interoperability and Conformance Test Framework 1.1, Oct. 2004.
[11] W3C, XML Path Language 2.0, June 2006.
[12] W3C, XML Query Language 1.0, June 2006.
[13] OASIS, ebXML Messaging Service Specification 2.0, April 2002.
[14] OASIS Simple Object Access Protocol 1.2, June 2003.
[15] eTSL, event-driven Test Scripting Language, Working draft, OASIS IIC Technical Committee, September 2006.
[16] RosettaNet Ready™, RossettaNet Self-Test Kit, December 2002.
[17] Web Service Interoperability Consortium, Web Service Testing Tools, http://www.ws-i.org/deliverables/workinggroup.aspx?wg=testingtools, June 2003.
[18] IEEE, Automatic Test Markup Language Draft, December 2004.
[19] RosettaNet, RosettaNet Implementation Framework 2.0, February 2000.
[20] W3C, Document Type Definition, December 1999.
[21] W3C, Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616), June 1999.
[22] IETF, Simple Mail Transfer Protocol (RFC 2821), April 2001.
[23] OASIS, ebXML Implementation, Interoperability and Conformance TC, Deployment Profile Templates V1.1, October 2006.