An Implementation Case of Business Centric Event-driven SOA Test Framework[1]

Youngkon Lee
e-Business Department, Korea Polytechnic University
2121 Jeongwangdong, Siheung city, Korea
yklee777@kpu.ac.kr

ABSTRACT: *This paper presents an implementation case study for business-centric SOA test framework. The reference architecture of SOA system is usually layered: business process layer, service layer, and computing resource layer. In the architecture, there are so many subsystems to affect system performance, moreover they relate with each other. As a result, in the respect of overall performance, it is usually meaningless to measure each subsystem's performance separately. In SOA system, the performance of the business process layer with which users keep in contact depends on the summation of the performance of the other lower layers. Therefore, measuring performance of the business layer includes indirect measurement of the other SOA system layers. We devised a business-centric SOA test framework in which activities and control primitives in business process managers are simulated to invoke commands or services in a test scenario. That is, in the test framework, a real business process scenario can be replaced to a mimicked business process test scenario, which is executed in a test proxy based on event mechanism. In this paper, we present the concept of business process activity simulation, 2-layered test suites model, and reference architecture.*

Keywords: SOA, BPM, test suites, event-driven

Received: 11 March 2009, Revised 14 April 2009, Accepted 29 April 2009

1. Introduction

Service Oriented Architecture (SOA) is generally defined as a business-centric IT architectural approach that supports integrating businesses as linked, repeatable business tasks, or services [1]. SOA enables to solve integration complexity problem and facilitates broad-scale interoperability and unlimited collaboration across the enterprise. It also provides flexibility and agility to address changing business requirements in lower cost and time to market via reuse.

SOA has a lot of promises of interoperability, however, at the cost of: lack of enterprise scale QoS, complex standards which are still forming, lack of tools and framework to support standards, and perform penalty. Recently, as SOA has been widely adopted in business system framework, performance issues in SOA are raised continuously from users and developers.

SOA system is generally composed of various subsystems, each of which relates intimately with others. Therefore, if performance issues are raised, it's very difficult to find out clearly what's the reason. For example, if a business process in SOA system has longer response time than before, there could be various reasons: cache overflow in a business processor, wrapping overhead in service interface, or exceptions in computing resources, etc. One thing clear is that the performance of business process layer depends on the lower layer and measuring the performance of business layer includes indirect measuring the performance of all the lower layers. But, most test frameworks developed until now focus on measuring SOA messaging performance, as we present in section 2. They almost adopt batch-style testing where all the test cases are executed in a sequence.

OMG published a standard SOA reference model, MDA (Model Driven Architecture) [2]. It is widely adopted in real world because it presents normative architecture and enables SOA system to be implemented in a business-centric approach. In the MDA, a business process is designed firstly in a way for satisfying business requirements and later services are bounded to

[1]This test framework has been implemented in an e-Government project sponsored by KIEC(Korea Institute of Electronic Commerce).

the activities in the business process. Business processes are described in a standardized language (e.g. WSBPEL) and they are executed generally on a business process management (BPM) system. Some subgroups in OMG are also studying actively how to apply event-driven architecture (EDA) on SOA system and they proposed a draft that EDA can be used as a bridge among SOA service groups because EDA provides a mechanism to decouple concretely services [3].

For testing SOA systems implemented according to the MDA reference model in business-centric way, test harness should have business process simulation functionality so that it may behave as BPM and at the same time test overall performance. This means that the test harness can execute business process, perform tests, and gather metric values.

We devised a new SOA test harness, **BOSET**[2], focusing on business process layer. It adopts a proxy mechanism, in which business processes and activities are simulated and executed to invoke events. The events initiate the service invocation so that the test system can gather the metric of the service performance. For the business-centric test execution, we also designed **test suite**, which is a document including structured and standardized test script. The test suite enables test harness to change its configuration flexibly according to the change of test target.

In section 2, we present some related works. Section 3 provides the principle requirement for test suite. In section 4, we describe the principle of test suite design. Section 5 presents briefly event-driven execution model and section 6 shows the related event data structure and operations in detail. Section 7 presents reference architecture for SOA test framework. Conclusions are presented in last section.

2. Related Works

There are various test frameworks and script languages developed or proposed for testing Web services systems, business processes, or business applications. This section briefs representative test systems and scripts.

2.1 Web Services Quality Management System

This system has been developed by NIA(National Information Agency in Korea) in order to measure Web services quality on the criteria of WSQM (Web Services Quality Model) quality factors [4]: interoperability, security, manageability, performance, business processing capability, and business process quality. This system contributes to consolidate the quality factors of SOA. However, it requires expanding its architecture to apply SOA system, because it targets to only Web services system.

2.2 ebXML Test Framework

This framework has been implemented by NIST and KorBIT for testing ebXML system according to OASIS IIC Specification [5]. It could test packaging, security, reliability, and transport protocol of ebXML messaging system implemented by ebMS specification [6]. The main purpose of it is to test conformance and interoperability of ebXML messaging system, so it is not proper to test service oriented systems. Besides, it cannot test ad hoc status resulting from various events, because it is not event-driven but batch-style test framework.

2.3 JXUnit and JXU

JXUnit [7] and JXU [8] is a general scripting system (XML based) for defining test suites and test cases aimed at general e-business application testing. Test steps are written as Java classes. There is neither built-in support for business process test nor support for the event-driven features. However, as a general test scripting platform that relies on a common programming language, this system could be used as an implementation platform for general e-business test.

2.4 ATML (Automatic Test Mark-up Language)

In its requirements, this specification provides XML Schemata and support information that allows the exchange of diagnostic information between conforming software components applications [9]. The overall goal is to support loosely coupled open architectures that permit the use of advanced diagnostic reasoning and analytical applications. The objective of ATML is focusing on the representation and transfer of test artifacts: diagnostics, test configuration, test description, instruments, etc.

[2] BOSET: Business Oriented SOA Execution Test Framework

2.5 Test Choreography Languages

These are standards for specifying the orchestration of business processes and/or transactional collaborations between partners. Although a markup like XPDL [10] is very complete from a process definition and control viewpoint, it is lacking the event-centric design and event correlation / querying capability required by testing and monitoring exchanges. Also, a design choice has been here to use a very restricted set of control primitives, easy to implement and validate, sufficient for test cases of modest size. Other languages or mark-ups define somehow choreographies of messages and properties: ebBP[11], WS-BPEL[12], WS-Choreography[13]. The general focus of these dialects is either the operational aspect of driving business process or business transactions, and/or the contractual aspect, but not monitoring and validation. Although they may express detailed conformance requirements, they fall short of covering the various aspects of an exhaustive conformance check e.g. the generation of intentional errors or simulation of uncommon behaviors. In addition, the focus of these languages is mainly on one layer of the choreography – they for instance ignore lower-level message exchanges entailed by quality of service concerns such as reliability, or binding patterns with the transport layer.

3. Requirements for Test Suite

Because SOA system is very complex and variable and has a number of heterogeneous subsystems, test suites including test logic and test cases should satisfy following requirements.

Event-driven and time-independent execution model: The test script must be executable either for real-time verification or as off-line (deferred) validation over a log of the interaction. Test cases also must be able to react to all sorts of events, and correlate past events. For these reasons, all input must be captured in the form of events and wrapped into a standard event (XML) envelope. The coordination of test-case executions within a test suite is also event-driven. The state of the test case workflow is also represented as events so that no additional persistence mechanism is required by a recoverable test engine.

Protocol-agnostic and platform-ubiquitous: Test script logic and control are abstracted from SOA protocols; it is versatile for messaging, business process, and business content testing regardless of technologies. Hence it can be used with either ebXML AS2 or Web Services message profiles. Of course a test case script that verifies business headers in ebXML may not apply to Web service messages, but a change in event-adapter should be the only modification needed to adapt a test script focused on verifying business transaction and payloads, from one message protocol to the other.

Adaptable interface: In our approach, the SOA test framework should have proxy which is delegated to replace temporarily BPM system. As a result, test framework has facilities to interface seamlessly services, functions, and components. For example, we implemented a service adapter, which transforms service appearance for adapting services. There could be plug-in systems which enable module or components to be easily connected and service wrappers which encompass functions in legacy systems into service types.

Extensible coverage of BPA simulated: BPA set simulated in test framework should be extensible to cope with the change of BPM systems which could be test target. Each BPA simulated should follow a standardized interface for connecting services.

4. Test Suite Design

Test suite means a document which describes the test target and test procedures. Test target is usually extracted from SOA standard specification. Test procedure could be used to control test flows. For making it easy, we designed 2-layered model for test suites: abstract test suites (**ATS**) and executable test suites (**ETS**) as shown in Figure 1. ATS describes test metadata of target expressed in test assertions and procedure and ETS describes executable test steps in the format of test execution language.

A test assertion is a testable or measurable expression for evaluating the adherence of part of an implementation to a normative statement in a specification. There is always a need to make explicit the relationship between a test assertion and the precise part of the specification to which it applies.

Test procedure describes test flow composed of a series of test activities which are simulated to business process activities. It is used in a test proxy, which is delegated as a process controller for test on replace of a BPM system. Test environment is a configuration description of a test harness.

ETS is a script for presenting each test step (in the other words, test case). It is independent from the SOA standard specification and domain environment but depends on the test execution model. For supporting machine and human readability, its format follows predefined XML schema and it has basic operation sets to initiate, control, and process events. Table 1 shows the basic operation sets in ETS.

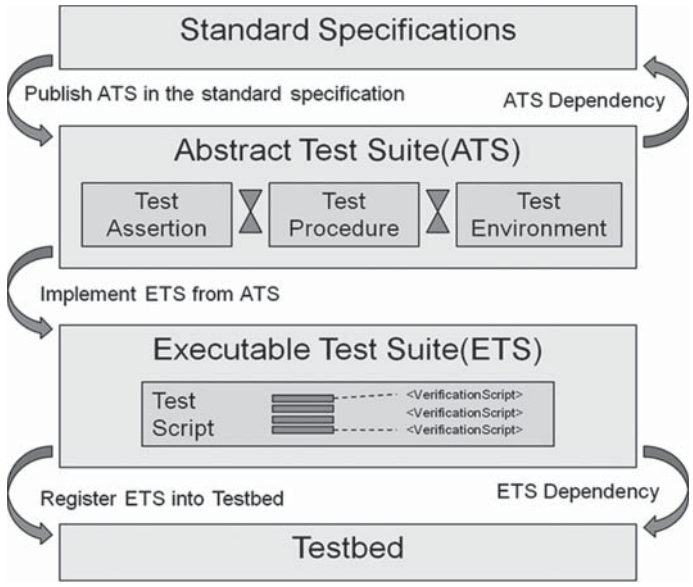

Figure 1. BOSET Test Suit Structure

Operation type	**Operation name**	**Description**
Event operation	*post*	generate an event
	find	select event(s) from EventBoard
	mask	mask or unmask some past events to a monitor instance
Monitor flow Control	start	start a new instance of a monitor
	set	assign a value or an XML infoset
	sleep	suspend an instance of a monitor
	cad	check-and-do operation.
	jump	pursue the execution thread at another (labelled) test step inthe monitor
External resources	call	invoke either an event-adapter or an evaluation-adapter
Test case control	actr	dynamically activate a trigger
	exit	terminate the current test case

Table 1. Basic Operations in ETS

5. Event-driven Execution Model (EDEM)

For invoking services, we adopted event-triggering mechanism according to business process activity. The event-triggering mechanism includes following concepts:

- **Event triggered by simulated BP**. An event is triggered by a business activity which is mimicked for execution in test proxy.
- **Services invoked by an event**. A service on a test platform is invoked by an event, which sends an initiating message synchronously. An initiated service can invoke the other services by replicating and propagating the message according to the test mission.
- **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 BOSET, is supported by an entity called Event Board.

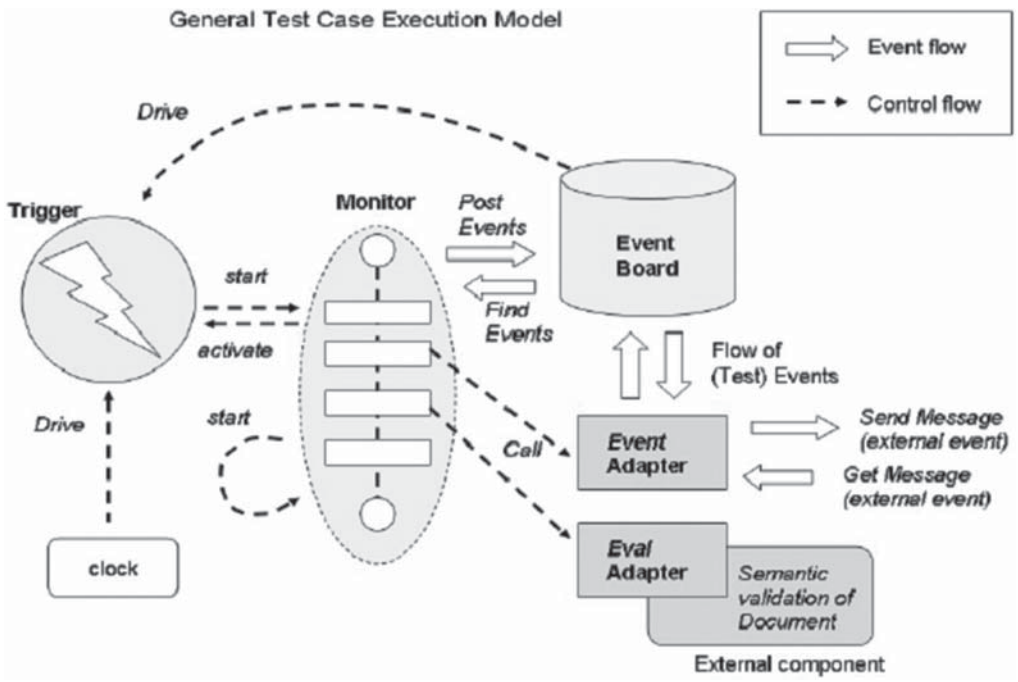

Figure 2. Event-Driven Execution Model

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

- **Messaging gateways.** Message traffic expected in all e-Business 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.** Agile verifications on business documents, ranging from schema validation to semantic rules over business content, are delegated to Evaluation-Adapters.

While these features may themselves be potentially complex, it has been possible in BOSET to identify a minimal set of controls sufficient for SOA testing. For example, workflow control only makes use of the simplest control primitives that have proved sufficient for test cases, not pretending to replicate the full range of workflow operators. Event correlation and querying rely on simple selection expressions based on XPath.

Based on the main concepts, the test execution model requires following components (Figure 3):

Monitor: This 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. The first monitor started for a test case (i.e. Started by a Trigger) is called root monitor for the test case. There is always a trigger at the origin of monitor(s) execution (directly or indirectly). 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) 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 to 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. The following figure illustrates the structure of a test suite:

Event (or Test 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 event board at the end of the CES they are associated with.

Event Board (EB): The event board provides event management functions, which invoke SOA services to operate by sending messages. 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. Data type and operations in the EB are presented in next section in detail.

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 SOA 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.

Evaluation Adapter: An evaluation 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 evaluation adapter is always invoked by a monitor. On invocation, an evaluation adapter returns an XML infoset summarizing the outcome, which can be evaluated later in the monitor workflow.

6. Event Board

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 as the “Event Board” (EB). Relying on event analysis for test cases also provides more control on the execution

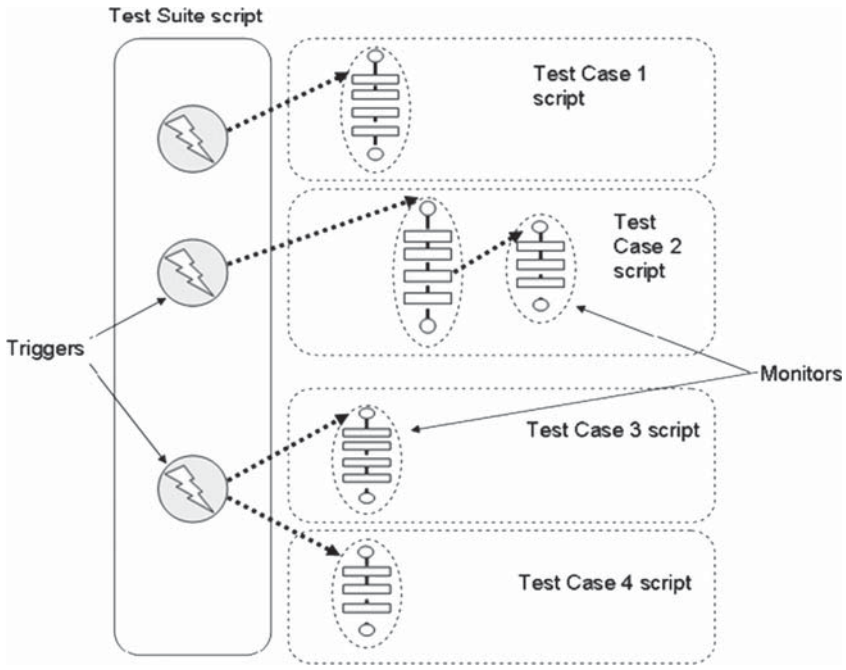

Figure 3. Test Case Triggering

time, which may be deferred. The notion of event or “test event” must be understood as an abstraction of any event that may impact the execution of a test case, or result from it, e.g. business messages received or sent, error notifications, outcome of a test case, test operator intervention, etc.

The EB also provides operations for managing events. A standard (XML format) representation of the invocations of such operations is proposed for the purpose of portability of test case scripts.

Although an implementation of the EB will likely act as an event sink, the EB is only assumed (1) to represent a log of events that have occurred, (2) to support an event model (or rather, an event-envelope model) and (3) to provide some form of management of and access to these events.

6.1 The Event Model

Every event posted to the EB is given an envelope, or event-envelope. An event envelope is an object that facilitates the event management. It has the following attributes:

- **iD:** a generated number assigned to an event at the time of its posting to the EB. The iD uniquely identifies the event. • • The iD defines a total order for the events in the board. This order is the same as the time order reflected by the timePost attribute.
- **timePost:** date/time of the event posting.
- caseId: the ID of the CES(case execution space) which the monitor instance that generated this event belongs to (in case the event is generated by a monitor).
- **mInstId:** the ID of the monitor instance that generated this event (in case the event is generated by a monitor).
- **evType:** event type, for which there are a few reserved values: “endmonitor”, “endcase”, “endsuite”.
- **temp:** a boolean that is “true” if the event is temporary.
- **evProperties:** a list of name / value pairs open for profiling.

The above attributes are “added” to the original event – or event content - for facilitating the processing of the event in the context of the board. The evProperties items are derived from the content of the original event, and represent an abstraction of it. It may contain references to external objects not managed by the EB. It may be used as a manifest for the actual event content. The event envelope is represented as an XML infoset of the form:

```
<define name=”Event”>
<element name=”event” datatypeLibrary= “http://www.w3.org /2001/
XMLSchemadatatypes”>
<attribute name=”id”><data type=”integer”/></attribute>
<attribute name=”timepost”><data type=”dateTime”/> </attribute>
<attribute name=”evtype”><text/></attribute>
<attribute name=”temp”><data type=”boolean”/> </attribute>
<optional>
<attribute name=”caseid”><text/></attribute>
<attribute name=”minstid”><text/></attribute>
<element name=”evproperties”>
<zeroOrMore>
<element name=”property”>
<attribute name=”name”/>
</element>
</zeroOrMore>
</element>
</optional>
<element name=”content”>
   <!-- a wrapper for the original event content -->
</element>
</element>
</define>
```

Table 2. XML Infoset for Event Envelop

The <content> element is a wrapper for any document associated with the event. For example, an event may be a SOAP message either sent or received. In that case the <content> element contains the SOAP envelope or a subset of it. In case there are attachments (MIME parts) these may remain external to the event envelope representation, and be referred to.

6.2 Event Board Operations

An operation, **Post**, adds an event entry to the event board (EB) – more precisely, adds an event envelope instance. Some attributes of the event envelope are automatically added by the test engine implementation (ID, timePost), others are implicitly passed or set (caseId, minstid). The event type @evtype needs be set for non-predefined events. The event @temp attribute is set to "true" by default. The <evproperties> and <content> elements must be explicitly defined in the post statement, if they must be added to the event envelope. All arguments of the post statement are optional.

```
<define name="Post">
<element name="post">
<optional>
<attribute name="step"><text/></attribute>
<attribute name="evtype"><text/></attribute>
<element name="evproperties">
<!-- as in event definition -->
</element>
</optional>
<element name="content"> <!-- event content -->
</element>
</optional>
</element>
</define>
```

Table 3. Infoset for Post Operation

An operation, **Find**, selects 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. To be eligible for selection by this operation, an event of the event board must satisfy the following conditions:

- the event is not masked for the monitor instance executing "Find",
- the event posting time (dateTime) is in the window [visibilityDate, time-cursor]
- In case it is a temporary event, it is associated with the same CES as this monitor instance (it has been posted by a monitor instance from the same test case execution).
- The event is within scope (if any, defined by <scope> argument)

Only such events will be considered for selection. If they satisfy the selector element (if any, defined by <selector> argument), then they will be part of the result set for the operation. The produced selection conforms to the <view> argument, if any, that allows for projecting only a subset of event data for each selected event, into the result representation.
The parameters have the following semantics:

- **tryDuration:** intuitively, how long can the Find operation "last" from the (virtual or real) time it starts to execute (called the "effectiveTime"), until the event Board (EB) is in a state where the operation Find returns a positive result (i.e. at least one selected event.) In other words, the Find operation may "query" the event board several times until it returns a positive result, over the time window [effectiveTime, effectiveTime + tryDuration]. The first positive result obtained during these attempts is the final outcome used in subsequent steps of the monitor execution. If tryDuration is absent: the find operation is executing only once, at effectiveTime (default value for tryDuration is 0).
- **scope:** specifies a set of events of which the event selection must be a subset. The returned set of selected events will be the intersection of those selected from the EB (satisfying the <selector> element) and those from the scope. The scope is specified either in the same way as the <selector> argument, or as the result of a precious selection. If not present, the default scope is the entire EB.
- **selector:** specifies a condition that each event to be selected from the EB must satisfy. May use XSLT2.0 path expressions, inside a <xpath> container element. [may use Xquery predicates, inside a <xquery> container element] The selector

applies to the sequence of events represented in the EB, that are in scope and also within the time window [visibiityDate, time-cursor] associated with the CES within which the selection occurs.

- **view:** determines how much data from each event record in the EB must be reproduced in the produced selection. It is specified as a set of Xpath expressions. If not present, only the event envelope data is returned, i.e. The event/content element is absent.

```
<define name=”Find”>
<element name=”find”>
<optional>
<attribute name=”step”><text/></attribute>
<attribute name=”tryduration”><data type=”duration”/> </attribute>
<element name=”scope”>...</element>
</optional>
<element name=”selector”>
<optional>
<attribute name=”get”>
<choice>
<value>first</value>
<value>last</value>
<value>all</value> <!-- default -->
</choice>
</attribute>
</optional>
[selection filter: choice of <xpath> or <xquery>]
</element>
<optional>
<element name=”view”>..</element>
</optional>
</element>
</define>
```

Table 4. Infoset for Find Operation

7. Reference Architecture

For testing SOA systems that have various components and flexible architecture, test requirement and the change in a test target should be rapidly applicable on a test harness. Thus, the test harness should reuse easily test components and be re-configurable.

BOSET is composed of a test component part and a test interface part (Figure 4). The test components include modules defined in EDEM, which are classified as stationary and non-stationary. Stationary module is static independent of any specific standard and/or test environment. Non-stationary module could be changed dynamically according to standards or test suite designs.

Stationary test components is composed of **TMC**(Test Main Component, Test Driver) and **TCE**(Test Configuration Engine). TMC orchestrates other test components and interfaces, and consequently drives the execution of test. TCE dynamically sets up test components in accordance with a configuration profile. **TSE**(Test Sequence Engine) interprets and drives executable test steps and interacts with test other components and interfaces.

Non-stationary test components include a test service module and an interpreter. Test Service stimulates target **SUT**(System Under Test) with pre-defined actions, which include instructions at the test state. The actions could be modified or created for the test specifics. Interpreter reads the test case and then parses it into test procedure, test assertions, and configuration information. Interpreter could be modified according to test suite design.
For interaction with SUTs, test drivers, and test users, BOSET has following interfaces:

- **MEI** (Messaging Engine Interface): delivers messages to/from SUTs based on the message protocol used. i.e., ebMS engine, SOAP engine, etc.
- **TVI** (Test Validation Engine): validates messages according to verification script. i.e, Xpath, Schematron, Xquery, JESS, OWL, etc.
- **TUI** (Test User Interface): provides user-interface using web or intranet. i.e, IIC web UI, WS-I UI, etc.

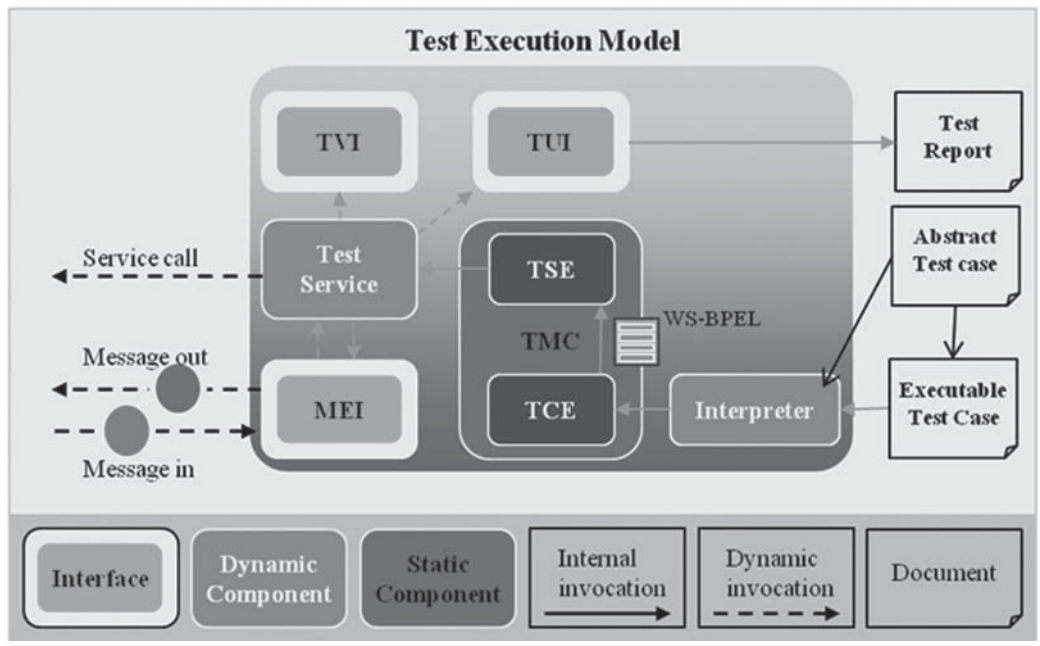

Figure 4. Reference Architecture for EDEM

Interface	Operation Name	Input	Output
Test Validation Interface(TVI)	Validation	Validation script and target messages	Validation result
Message Engine Interface(MEI)	Sending	Message	Message Log
	Query (Receiving)	Query Script	Message Log
Test User Interface(TUI)	Reporting	Results	Report Document

Table 5. Abstract Definition of Test Interfaces

For interface model for Service Description, we adopted WSDL (Web Service Description Language), a standard specification. TCE discoveries and dynamically deploys interface modules in the Universal Test Module Repository. Configuration document could be registered in a registry implemented according to UDDI (Universal Description, Discovery and Integration) specification. TSE orchestrates deployed test component and interface modules. For dynamic invocation, WS-BPEL (Web Service Business Process Execution Language) primitives could be used.

8. Conclusion

We presented a SOA test framework, which has been implemented in Korea government side for testing public SOA systems. The framework facilitates to test SOA systems by introducing the concept of business activity simulated event proxy. For the framework, we also devised 2-layered test suites: abstract test suite and executable test suite. The abstract test suite describes test workflow based on a business process. The executable test suite represents test operations in detail for test case execution. This model decouples test procedure and test cases; as a result it enhances the reusability of test components. We also provide reference architecture for SOA test framework, which will be a guideline to later implementation of business-centric test framework.

References

[1] Nickul, D. (2007). Service Oriented Architecture (SOA) and Specialized Messaging Patterns, Adobe technical paper, Dec.

[2] Miller, Joaquin., Mukerji, Jishnu (2003). MDA Guide Version 1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf, OMG, June.
[3] Covington, Robert D (2006). OMG EDA Standard Review, http://www.haifa.il.ibm.com/Workshops /oopsla2006/present/omg_eda_rfi _presentation.pdf
[4] Lee, Y et al., (2008). Web Services Quality Model 1.1, OASIS WSQM TC, Oct.
[5] Durand, J. et al., (2004). ebXML Test Framework v1.0, OASIS IIC TC, Oct.
[6] Wenzel, Peter et al., (2007). ebXML Messaging Services 3.0, OASIS ebMS TC, July.
[7] Java XML Unit (JXUnit), http://jxunit.sourceforge.net.
[8] JUnit, Java for Unit Test, http://junit.sourceforge.net.
[9] ATML, (2006).Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML, IEEE, Dec.
[10] XPDL: (2003).XML Process Defi nition Language) (Workfl ow Management Coalition) Document Number WFMC-TC-1025: Version 1.14 October 3.
[11] OASIS, (2006). Business Process Specifi cation Schema 1.0.1, May 2001 and ebBP, v2.0.4, October.
[12] OASIS, Web Services Business Process Execution Language 2.0 (committee draft in review phase), August 2006.
[13] Web Services Choreography Description Language (WSCDL), (2005).Version 1.0, (candidate recommendation) November.

Author bibliography

Youngkon Lee is a professor of e-Business department at Korea Polytechnic University in Korea. He received his Ph.D. from Korea Advanced Institute of Science Technology. His major research interests include software and service quality of SOA, semantic web service, intelligent service framework and business process automation. He is a individual member of OASIS and attends TAG and TAMIE technical committee. He is also advising a Korea e-government project that is innovating overall government IT framework.