A Web Service-based Reconfigurable Testbed for Business-to-Business (B2B) Integration

Buhwan Jeong, Jungyub Woo, Hyunbo Cho
Department of Industrial and Management Engineering
Pohang University of Science and Technology (POSTECH)
San 31, Hyoja, Pohang, 790-784, South Korea
{bjeong, woo, hcho}@postech.ac.kr

Boonserm Kulvatunyou, Nanad Ivezic
National Institute of Standards and Technology
MS 8263, Gaithersburg, MD 20899, USA
{serm, nivezic}@nist.gov

Jaeman Lee
Korean Agency for Technology and Standards
Joongang, Kwachun, 427-716, South Korea
ljm@ats.go.kr

Abstract

Several types of standards and corresponding software solutions have been created to enable business-to-business (B2B) integration. Testing for conformance and interoperability among these solutions are inter-dependent. This signifies the needs to make testing more efficient and to save testing facility development costs. Consequently, a reconfigurable testing facility that is capable of expediting conformance and interoperability tests for multiple types of B2B solutions is desirable. This paper outlines a framework for a reconfigurable testbed to verify B2B compliance using web services technologies. Test components of the testbed are designed as web services that are discoverable, plug-and-playable and composable. The framework uses a web service orchestration language, i.e., BPEL4WS, to configure the testbed. Case studies of using the approach to configure testbeds for testing of XML Schema design compliance to organizational best practices and for testing of information mapping demonstrates the effectiveness of the framework.

1. Introduction

As the e-business, especially business-to-business (B2B), has matured, it requires more software solutions to support its transactions. This has unfortunately intensified the need for interoperability among those solutions. This problem is inevitable because 1) the market requires various types of software, 2) each type of software is subjected to various standard specifications and versions, 3) software solutions implement the same standard with varying assumptions (e.g., industry domain), and 4) the running environments and platforms are different from each other. A notable approach to achieve interoperability among them is to verify and certify every solution participating in the integration in terms of both conformance and interoperability. The prerequisite to this approach is a test framework that enables uniform and consistent verification of virtually any kind of B2B software solutions.

The paper outlines a framework for a reconfigurable testbed. It defines test components that are modularly designed as web services. Using web services, test components are plug-and-playable enough to dynamically compose a testbed according to the type of solutions to be tested. Definitions of such test components and their configurations are provided as part of the research.

Next, we briefly introduce backgrounds on B2B solutions testing and web services technologies. Then, the reconfigurable testbed framework is addressed, which is followed by web service definitions and configurations of test components. Subsequently, a case study is provided before we finally give the conclusion and future plans.

2. Test framework and web services technologies

There are two leading initiatives in B2B solutions testing – the WS-I (Web Services Interoperability) organization and the OASIS (Organization for the Advancement of Structured Information Standards) technical committee for ebXML IIC (Implementation, Interoperability, and Conformance). The WS-I is geared more specifically to promote web services interoperability by providing guidance, recommended practices, and testing tools. Currently, it delivers implementation guidelines for how web services specifications should be used together for best interoperability; sample web services

applications compliant with the guidelines; and test tools to monitor the messages exchanged and to analyze the resulting logs [1].

The OASIS IIC technical committee proposed a test framework (hereafter, IIC Test Framework) to verify any ebXML-compliant solutions. Initially, the IIC Test Framework supports testing of ebXML MSHs (Messaging Service Handlers) only; however, the latest draft includes other functions to validate the SOAP (Simple Object Access Protocol) binding. This is a provision to support testing of web services. Because we will extend the IIC Test Framework for the proposed reconfigurable testbed, more detail about the framework will be described.

The IIC Test Framework is flexible to perform both **conformance** and **interoperability** tests. Its main components are **test driver** and **test service**. The framework also defines a **test suite** specification. The test driver is responsible for interpreting test cases and driving test execution. The test service represents the software layer beyond the target layer to be tested. The test suite is a collection of executable and referential test materials, which include test requirements, functional requirements, test assertions, test profiles, test cases, test message data, etc [2].

The service-oriented architecture (SOA) is a paradigm for building distributed applications from a collection of services. Each service is considered as a self-contained, self-describing, modular component. A service has well-defined interfaces from which the service can be obtained through XML messages. The SOA using web services technologies enables loosely coupled applications integration within and across enterprises using the Web infrastructure. Web services can be discovered, selected, composed, and invoked. WSDL (Web Service Description Language) and the UDDI (Universal Description, Discovery, and Integration) registry support the service advertisement and discovery. Recent researches in this area have shifted to semantics-aware advertisement and discovery. Either WSDL or UDDI is extended to be capable of containing semantics-rich information [3]. The process to bring several independent web services together and form a new service is called a web services composition. The traditional approach to web services composition is static via manually encoding a workflow specification. Recent approaches focus more on automatic composition using AI-based planning [4].

There are two schemes for representing a workflow in the web services composition. The first scheme uses the concept of ‘orchestration’. The web services orchestration has a web service that acts like an orchestral conductor. This orchestral service controls the overall collaboration procedures. The typical orchestration language is BPEL4WS (Business Process Execution Language for Web Services), which inherits features from both WSFL and XLANG. The other scheme uses the concept of ‘choreography’. The web services choreography captures collaborations among web services. While the orchestration explicitly defines relationships between the conductor service and others, the web services choreography describes the observable behaviors of each web service. That is, a single choreography describes only one partner’s participation in a message exchange. WSCI (Web Service Choreography Interface) is a workflow specification adopting this scheme [5].

3. Testbed architecture

3.1 Reconfigurable testbed framework

Figure 1 illustrates the testbed architecture, which basically inherits several concepts from the IIC Test Framework such as the test driver and the test service. Even though the IIC test driver is responsible both for executing test cases according to test steps and for validating test messages, the new test driver only executes test steps. In other words, it has a generic test execution function only. The test message validation function is separated to a test validation component. This separation allows the test driver to connect to various types of validation components consequently allowing it to use various test script languages (e.g., XPath [9], XQuery [10], Schematron [12], Java Expert System Shell – JESS [14]) to capture test assertions. The other new and notable component is the test case translator. Each translator converts abstract test cases, which are specific to types of B2B software solutions, into a common executable test case understandable by the test driver. The test driver connects to an appropriate translator and one or more test validation components to form a testbed.

Comment [B.S.K1]: Add references here.

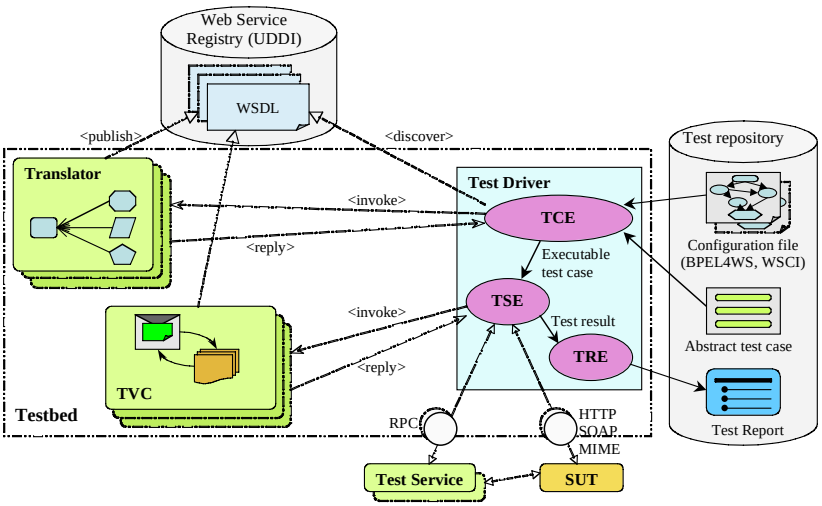

Figure 1. Web service-based testbed framework

Test driver is like the brain of the testbed. It is a test component that reads and interprets a test case, orchestrates other components, and drives the execution with the SUT(s). The typical procedure for executing a test case or a test suite is: 1) read a configuration file and discover and set up test components via its test configuration engine; 2) take an abstract test case and transform it into an executable test case via a discovered translator; 3) initiate test and send (receive) test messages to (from) the target SUT; 4) validate received messages via a discovered TVC; and finally 5) generate a test report from validation results. This procedure indicates that the test driver has three modules including the **test sequence engine** (**TSE**), the **test configuration engine** (**TCE**), and the **test report engine** (**TRE**). The TSE drives the execution by communicating with the SUT(s) and/or test service(s) via transport and service adapters, respectively. Proper adapters are indicative in the executable test case. The TCE is responsible for dynamically setting up the testbed according to a configuration file (e.g., BPEL4WS, WSCI). Depending on the type of SUT and assertion scripts indicated in the abstract test case, the TCE will discover and compose the best suitable test case translator and TVEs. The TRE aggregates test results from several test executions and generates a test report.

Test validation component (TVC) verifies messages according to validation assertions. An assertion may be scripted in languages such as XPath, Schematron, XQuery, JESS, OWL, and so on. The TVC should be independent of the TSE, so that when a new scripting language is adopted, a new TVE can easily be plugged in the testbed. Consequently, TVCs should be developed as web services.

Test case translator is a test component that converts an abstract test case into an executable test case representation. The abstract test case is specific to the type of the target specification. An executable test case is a set of step-by-step instructions understood by the TSE. This common representation allows more reuse of the generic TSE, while specific abstract test case representations make it easier for solution domain experts to prescribe test cases. The translators will be developed as a web service.

In addition, the testbed requires a **test repository** to persistently store all the artifacts including configuration files, test cases, test messages, and test results. The **test services** may also be implemented as web services.

The use of web services for TVCs, translators and test services results in an agile testbed. Several test components can be reused for an emerging B2B specification. The specification expert can focus on defining how and what a software implementing the specification should be tested using his/her familiar terms in the abstract test case representation. The specification expert can also use the assertion scripts that fit the need of the specification. The testbed administrator can then help build a test case translator that maximizes the reuse of the test components.

3.2 Test cases

In the reconfigurable test framework, test cases are layered into **abstract test cases** and **executable test cases**. Before executions, abstract test cases must be first translated into executable ones via appropriate translators.

The abstract test case is a high-level test case that captures test scenarios and assertions in a way specific to the target specification. The purpose is for the test case to be easily specified by the specification expert and easily understandable by the solution provider (who needs to perform the test). That is, those users do not have to understand the details of the testbed to write and execute the test. Another reason for the abstract test case is to cope with the no-single-fit-all situation. Various B2B specifications require varying information in a test case. For example, some types of test cases need communication protocol, operation end-point, message binding, and/or security configuration in the test case, while some only require such information specified at the execution or do not require at all.

On the other hand, an executable test case is directly consumable by the test driver. It is common across types of specifications. There are three possible actions in an executable test case similar to in the IIC test case. They are 'PutMessage', 'GetMessage' and 'TestAssertion'. The TSE follows these actions to drive the test. For example, an abstract test case may capture a test scenario in a single statement which implies three executable steps – put, get, and assert. A translation would segregate the single statement into three discrete steps understood by the TSE.

4. Web services for test component

4.1 Web services definition

Table 1 describes web service definitions of the test components including their abstract operation names, inputs and outputs. The test driver service on the first row is an entry point for test execution. It forms a testbed and provides its testing capability by dynamically invoking other web services including a translator, TVCs, and test services.

Table 1. Web service definitions for test components

Component	Operation Name	Input	Output
Test Driver	Test	Abstract test case and BPEL4WS	Test report
Translator	Translate	Abstract test case	Executable test case
TVE	Validate	Validation scripts and test message	Validation results
Test service	Command	Testing Type and Action name	Message received by SUT

4.2 Test component publication

WSDL [6] is a standard means of defining web services by indicating end-points (service address), messages, port types (operation definition), and operation binding (how to access the operation). WSDL is advertised via a central registry supported by the UDDI specification [7], through which a service requestor can subscribe and discover web services. Figure 2 illustrates a WSDL fragment of the validation service description, which includes message types and abstract definitions (i.e., *types* and *message*), operations (i.e., *portType* and *operation*), transport protocol (i.e., *binding*), and operation and service end-points (i.e., *service*). The message definitions, which reflect the inputs and outputs listed in Table 1, are externally defined and imported. In addition, it should be noted that partner link types (i.e., *plnk:partnerLinkType*) are specific to the proposed reconfigurable test framework. It is added for the purpose of assisting in the web services composition. In the example, it indicates that this web service is a TVC. Other components are defined similarly using the WSDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions …namespaces…>
 <types>
  <xsd:schema>
   <xsd:import location="ValidationRule.xsd"/>
   <xsd:import location="TestResult.xsd"/>
   <xsd:import location="TestMessage.xsd"/>
  </xsd:schema>
 </types>
 <message name="GetValidationRule">…</message>
 <message name="GetTestMessage">…</message>
 <message name="GetTestResult">…</message>
 <portType name="UploadTestMessagePT">
```

```
  <operation name="Upload">
   <input message="tns:GetTestMessage"/>
  </operation>
 </portType>
 <portType name="ValidateTestMessagePT">
  <operation name="Validation">
   <input message="tns:GetValidationRule"/>
   <output message="tns:GetTestResult"/>
  </operation>
 </portType>
 <binding name="TestMessageSoapBinding" …/>
 <binding name="ValidationRuleSoapBinding" type="defs:ValidateTestMessagePT">
  <soap:binding style="document" transport="…"/>
  <operation name="Validation">
   <soap:operation name="…" soapAction="…"/>
   …
  </operation>
</binding>
 <service name="TestValidationService">
  <port name="TestValiationPort" binding="tns:ValidationRuleSoapBinding"/>
 </service>
 <plnk:partnerLinkType name="tns:validatorPartnerLT">…
 </plnk:partnerLinkType>
</definitions>
```

Figure 2. Part of a TVE WSDL service description

4.3 Test components composition

BPEL4WS [8] is an implementation of the web services orchestration. It is used to configure a testbed. The configuration file is created off-line manually. It must satisfy the technical requirements induced from the abstract test case specification (and the type of target B2B software and specification). The technical requirements indicate functionalities and procedure needed to form a testbed. From those technical requirements appropriate test components are discovered or constructed if not exist and the test procedure is realized. The BPEL4WS captures the description of these necessary test components and test procedure.

The construction of a configuration file follows the general procedure for defining a process in the BPEL4WS by specifying 1) test component web services to be plugged in; 2) input and output messages for each service; 3) communication and invocation sequence; and 4) fault and event handlers.

In first step, the test component services are declared using *partnerLink*. Named partners are defined, each of which is characterized by the *partnerLinkType* in the WSDL. Additionally, *myRole* and *partnerRole* attributes indicate respective roles that the orchestrator and the partners will play.

The second step lists variables (i.e., *variable*) used to specify input and output messages for each service. In this step, *correlation*s are optionally stated to designate alternative messages for each message.

The third and most important step specifies activities (using *receive*, *invoke*, *reply*, etc) and their sequence (using *sequence*, *switch*, *while*, *pick* and *flow*). As part of the process, the configuration may also have error-handling and roll-back mechanisms (i.e., *faultsHandler*, *compensationHandler*). This is specified last.

Figure 3, for example, demonstrates the fragmentation of a sample testbed configuration file. The testbed has three components (i.e., TestDriver, Translator, and TVC) and consists of ordered activities of receiving an abstract test case (i.e., *receive*), converting it into an executable test case (i.e., *invoke* Translator), and validating test messages (i.e., *invoke* TVC with 'Upload' and 'Validation' *operation*s), and consequently reporting the result (i.e., *reply*). It is noted that the test driver (which contains the TSE and TCE) is used as the orchestral conductor service in this configuration, and it is also implemented as a web service.

```
<?xml version="1.0" encoding="UTF-8"?>
<process …namespaces…>
 <partnetLinks>
  <partnerLink name="TestDriver" …
               myRole="configurator"/>
  <partnerLink name="Translator" … />
  <partnerLink name="TVC"
      partnerLinkType="tss:validatorPartnerLT"
      myRole="validator"
      partnerRole="configurator"/>
```

```
</partnerLinks>
<variables>
  <variable name="abstractTestCase" … />
  <variable name="executionTestCase" … />
  <variable name="validationRule"
            messageType="defs:ValidationRule"/>
  <variable name="testMessage" … />
  <variable name="testResult" … />
  <variable name="testReport" … />
 </variables>
<sequence>
  <receive partnerLink="TestDriver" … />
  <invoke partnerLink="Translator" … />
  <invoke partnerLink="TVC"
          portType="tss:UploadTestMessagePT"
          operation="Upload"
          inputVariable="testMessage"/>
  <invoke partnerLink="TVC"
          portType="ValidateTestMessagePT"
          operation="Validation"
          inputVariable="valdationRule"
          outputVariable="TestResult" />
  <reply partnerLink="TestDriver" … />
 </sequence>
</process>
```

Figure 3. Part of exemplary testbed configuration in BPEL4WS

5. Case study

Two case studies are described in this section. The demonstration is the reuse of test components to compose information mapping testbed and the schema design quality testbed.

5.1 Information mapping test

As a case study, we apply the proposed reconfigurable testbed framework to create a testbed corresponding to the information mapping testbed. The aim of the information-mapping test (IMT) is at verifying that a software solution correctly maps information from its proprietary representation to the standard-based data exchange representation, and vice versa [11]. The test consists of two test subtypes including the input test and the output test. The latter output test verifies that the solution successfully reads and interprets the local content using a standard data exchange representation. Conversely, the input test verifies that the solution properly processes the content in the standard representation to its local representation.

Figure 4 depicts the system view of the IMT testbed made up of a test driver, a translator and an XSLT engine. We have implemented three additional web services: the translator the XSLT engine (as TVC) and the mapping test service, while the test driver is discovered and reused. During the execution, the test driver component initially reads a BPEL4WS configuration file and invokes these services. Among them, the translator service is invoked by the TCE and the others are invoked by the TSE. The mapping test service implements two operations including the input testing and the output testing.

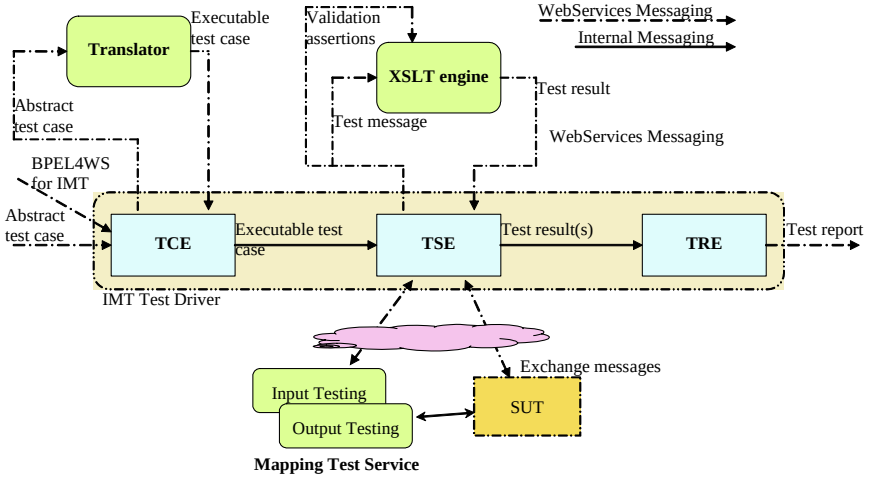

Figure 4. System view of IMT testbed

For demonstration purpose, we depict here a sample WSDL file fragment for the mapping test service as shown in Figure 5. In the same way, other services are also described in WSDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions …namespaces…/">
 <types … />
 <message name="requestMessage"/>
 <message name="receivedMessage"/>
 <portType name="MappingTestServicePT">
  <operation name="OutputTesting">
   <input message="requestMessage"/>
   <output message="receivedMessage"/>
  </operation>
 </portType>
 <binding name="MappingTestServiceBinding"
          type="MappingTestServicePT">
  <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/
        soap/http"/>
  <operation name="OutputTesting">
   <soap:operation style="document" soapAction="http://www.korbit.org/testbed/IMT?outputTesting">
    <input><soap:body use="literal"/></input>
    <output><soap:body use="literal"/></output>
   </soap:operation>
  </operation>
 </binding>
 <service name="OutputTestingService">
  <port name="MappingTestServicePort"
        binding="MappingTestServiceBinding"/>
 </service>
 <plnk:partnerLinkType name="MappingTestServiceLT">
  <role name="OutputTesting">
   <portType name="MappingTestServicePT"/>
  </role>
 </plnk:partnerLinkType>
</definitions>
```

Figure 5. Mapping test service description in WSDL

As shown in Figure 6, a BPEL4WS description allows those services to be composed from the test driver perspective as an orchestral conductor. The figure illustrates that the test driver selectively invokes an operation of mapping test service according to the property ‘testingType’ of the ‘IMT test case’.

```
<?xml version="1.0" encoding="UTF-8"?>
<process …namespaces…>
 <partnetLinks>
  <partnerLink name="TestDriver" … />  <partnerLink name="XSLTEngine" … />
  <partnerLink name="MappingTestService"
   partnerLinkType="MappingTestService LT" … />
  …
 </partnerLinks>
 <variables … />
 <sequence>
  <receive partnerLink="TestDriver" … />
  <invoke operation="Translation" … />
  <switch … >
   <case condition="getVariableProperty
       (tesingType) = outputTest">
    <invoke partnerLink = "MappingTestService"
            operation="OnputTesting" … />
   </case>
   <case condition="getVariableProperty
       (testingType) = inputTest">
    <invoke partnerLink ="MappingTestService"
        operation="InputTesting" … />
   </case>
</switch>
  <reply partnerLink="TestDriver" … />
 </sequence>
</process>
```

Figure 6. IMT testbed configuration in BPEL4WS

5.2 Schema design quality test

To show the proposed framework is reconfigurable depending on the type of testing, we applied it to create another testbed corresponding to the schema design quality (QoD) testbed. The purpose of the QoD test is for ensuring the XML schema compliance to organizational best practices (also called Naming and Design Rules (NDR) [12]. A QoD test case consists of validation rules specified in different test script languages including Schematron, JESS, and XSLT. In short, Schematron is an XML representation of rules based on XPath/XSLT expressions, the XSLT by itself is a procedural language, while JESS is a first order representation based on the CLIPS [15] syntax. These languages have their own strengths. The Schematron and XSLT being an XML native language makes it easier to encode validation scripts against an XML schema, while JESS provides more expressivity due to the capability to embed native Java classes and methods. The Schematron's data structure consists of assertion and then message. Hence, it is simpler to capture rules than the XSLT.

Comment [B.S.K2]: Need reference here too.

As shown in Figure 7, the QoD testbed has similar configuration to the IMT testbed, with an exception that it has two addtional validation engines (i.e., the Schematron engine and the JESS engine). We have implemented three additional web services: the translator, the Schematron engine and the JESS engine, while the test driver and the XSLT engine are discovered and reused. The test driver selectively invokes the Schematron, JESS, or XSLT engine according to the test script language used. Since the QoD test is not to verify a software solution, it does not need a test service. It is noted that, depending on the configuration files stored in the test registry, the user can agilely re-configure the testbed for other types of testing.

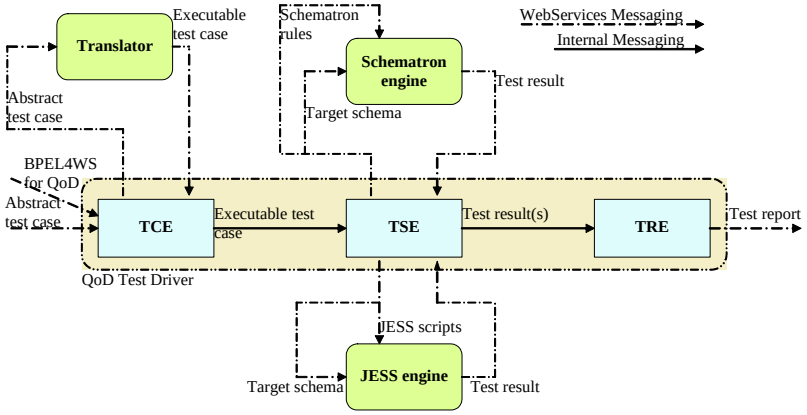

Figure 7. System view of QoD testbed

For demonstration purpose, we depict here the sample WSDL for Schematron engine service and BPEL4WS files as shown in Figure 8. The figure (B) illustrates that the test driver selectively invokes a validation engine according to the property 'ruleType' of the 'validationRule' variable (or message).

(A)

```
<definitions …namespaces…/">
 <types … />
 <messages … />
 <portType name="SchematronEnginePT" … />
 <service name="TestValidationService">
  <port name="SchematronEnginePort"
        binding="SchematronEngineBinding"/>
 </service>
 <partnerLinkType name="SchematronEngineLT">
  <role name="TestValidation">
   <portType name="SchematronEnginePT"/>
  </role>
 </partnerLinkType>
</definitions>
```

(B)

```
<process …namespaces…>
 <partnetLinks>
  <partnerLink name="SchematronEngine"
     partnerLinkType="SchematronEngineLT" … />
 </partnerLinks>
 <sequence>
  <receive partnerLink="TestDriver" … />
  <invoke operation="Translation" … />
  <invoke operation="Upload" … />
  <switch … >
   <case condition="getVariableProperty
      (validationRule, ruleType) = Schematron">
    <invoke partnerLink="SchematronEngine"
            operation="Validation" … />
   </case>
   <case condition="getVariableProperty
       (validationRule, ruleType) = Jess">
    <invoke partnerLink="JessEngine" … />
   </case>
  </switch>
  <reply partnerLink="TestDriver" … />
 </sequence>
</process>
```

Figure 8. (A) Schematron service description in WSDL and (B) QoD testbed configuration in BPEL4WS

6. Conclusion

The paper has shown a light-weighted reconfigurable testbed by incorporating the concept of web services. Each test component is implemented as a web service, so that the BPEL4WS can be used to bind them according to the target applications and test script languages. Each component is described in the WSDL. They are combined into a testbed via the BPEL4WS. The use of web services makes the testbed agile enough to test diverse applications and versatile enough to use various test script languages.

Future works include building an intelligent test driver that can autonomously configure a testbed by taking choreographies among web services (i.e., test components) and then dynamically discovering and composing those services. The precondition to this is that the web services are described in a semantic rich manner. In addition, for more robust testing, we need to closely investigate the test suites and further regulate such standard message formats exchanged among test components.

Disclaimer

Certain commercial software products are identified in this paper. These products were used only for demonstration purposes. 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.

Reference

[1] WS-I Organization Website (Accessed December 2005), Online available via <http://www.ws-i.org/>

[2] OASIS ebXML Implementation, Interoperability and Conformance (IIC) TC Website (Accessed November 2005), Online available via <http://www.oasis-open.org/committees/tc_home.
php?wg_abbrev=ebxml-iic>

[3] J. Jang, B. Jeong, H. Cho, and J. Lee, "Capability and Extension of UDDI Framework for Semantic Enterprise Integration", In *Proceedings of International Conference on Advances in Production Management Systems*, Rockville, MD. September 2005

[4] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods", In *Proceedings of 1st International Workshop on Semantic Web Services and Web Process Composition* (*SWSWPC 2004*), San Diego, CA. July 2004

[5] C. Peltz, "Web Service Orchestration and Choreography: A Look at WSCI and BPEL4WS", *WSJ Feature*, July 2003, pp.1-5

[6] Web Service Description Language (WSDL) Website (Accessed January 2006), Online Available via <http://www.w3.org/2002/ws/>

[7] Universal Description, Discovery and Integration (UDDI) Website (Accessed December 2005), Online Available via <http://www.uddi.org>

[8] T. Andrew et al., *Business Process Execution Language for Web Service* (BPEL4WS, *V. 1.1*), Online Available via <http://www-128.ibm.com/
developerworks/library/specification/ws-bpel/> (Accessed January 2006)

[9] J. Clark and S. DeDose, XML Path Language (XPath) Version 1.0, Online Available via <http://www.w3.org/TR/xpath>, November 1999

[10] W3C XML Query, Online Available via <http://www.w3.org/XML/Query/> (Accessed January 2006)

[11] B. Kulvatunyou, N. Ivezic, and A. Jones, "Content-level Conformance Testing: An Information Mapping Case Study", In *Proceedings of Testing of Communicating Systems: the 17th IFIP TC/6WG6.1 International Conference (TestCom 2005)*, Montreal, Canada, June 2005

[12] B. Kulvatunyou, N. Ivezic, and B. Jeong, "Testing Requirements to Manage Data Exchange Specifications in Enterprise Integration – A Schema Design Quality Focus", In *Proceedings of the 8th World Multi-Conference on Systemics, Cybemetics, and Informatics* (*SCI 2004*), Orlando, FL, July 2004

[13] R. Jelliffe, *The Schematron Assertion Language* (*V. 1.5*), Academia Sinica Computing Center, Online Available via <http://www.ascc.net/xml/resource/
schematron/Schematron2000.html> (Accessed December 2005)

[14] E. Friedman-Hill, *JESS: The Rule Engine for the Java™ Platform* (*V. 7.0b5*), Online Available via <http://herzberg.ca.sandia.gov/jess/> (Accessed January 2006)

[15] CLIPS, A Tool for Building Expert Systems, Online Available via <http://www.ghg.net/clips/
CLIPS.html> (Accessed December 2005)