# Test Requirements of Service Connections with their Applications to a Testing Tool

**Hoijin Yoon**,

Department of Computer Engineering,
Hyupsung University,
Hwaseong-si, Gyeonggi-Do, Korea

**Summary**
This paper presents what should be test requirements of service connections in Service-oriented architecture (SOA), and then it applies the test requirements to real SOA systems and a common web service test tool, SoapUI. Services in SOA are deployed on their proper layers according to whether they are biased on business process or application solution. An interaction between services in the different layers is called connection while an interaction between them in the same ones is called integration. So a connection is a new and important spot to be tested in SOA. This paper present test requirements for the connection and generates test requirements for SOA systems of real companies as the examples. And it also explains how to apply the test requirements to a common web service test tool, since many SOA developers still stay on the tools supporting Web services.

***Key words:***
*Service-oriented architecture, testing, service connection, web service, business service layer, application service layer, SoapUI*

## 1. Introduction

The major impact for the Service-Oriented Architecture (SOA) initiative is solving the age-old problem of integration. According to many analyst estimates, up to 30% of a typical IT budget is allocated to integration activities [1], which is directly related to process integration, enterprise integration, and mergers and acquisitions (M&A) integration. With the aid of SOA, resources can be shifted to more strategic projects, which otherwise would have been spent for the integration. SOA supports the demand for IT and process integration. In building an enterprise system as a SOA, there are two things we should consider as followed, and this paper proposes one of the solutions for them.

First, a connection between SOA services is a new spot to be tested in SOA. An interaction between services in the different layers is called *connection* to distinguish it from conventional integration. *Connection* is necessary in SOA, since SOA put services in their proper layers to keep the coupling of a business process and its applications loose. This layering system is one of the most important principle to support the loosely-coupled between business process and application solution, which is the main aim of many enterprise systems' going to SOA. A connection means interactions between a business service and an application service. Meanwhile, organizations are interested in upgrading their systems to SOA architectures, but they have been trying this still as pilot projects. The try could not be progressed to a real project. One of the main reasons is that there aren't any testing methods covering the connection's characteristics, even though almost every SOA system should request a high reliability as an enterprise-level system. In HP Software Universe 2007 [2] held at the beginning of 2007, many analysts emphasized that the success in building SOA architectures could critically depends on the connection testing. Although some testing techniques [3,4,5] have been developed for the web service integrations based on the first generation web service techniques such as SOAP, WSDL, and UDDI, they do not emphasize where business services and application services are deployed separately. Of course, some ESB products-IBM Web Sphere™ [6] or Fiorano BIS™- have implemented testing tools inside the products. But the tools still ask the test engineers to fill blank fields of the tools with test requirements and their test cases.

Second, many SOA developers still stay on the tools supporting Web services. In terms of testing, they rely on some popular open source products, such as *SoapUI, JUnit, and TESTMAKER*, which were developed for testing Web services. The tools could work well also on SOA, because the connection is technically still working through SOAP message exchanges as in Web service integrations. This paper proposes what should be included in their test steps of the connections according to the definition of the connection of services with supporting SOA principles.

From the considerations just mentioned before, this paper proposes what should be test requirements for connections of SOA services, and it defines test requirements of two real enterprise system examples. Once the test requirements are defined, this paper explains which part of *SoapUI* covers the test requirements.

Section 2 explains the service connection of SOA compared to the integration of Web services and then analyzes some existing testing tools to show why this paper chooses *SoapUI* as a tool. Section 3 defines a diagram to show layers and connections of SOA, and generates test requirements from the diagram. Section 4

Manuscript received March 5, 2010
Manuscript revised March 20, 2010

applies our approach to the two different enterprise cases of SOA systems, and then it shows how to use the test requirements in *SoapUI*. Section 5 concludes this paper with contributions and future work.
From this section, input the body of your manuscript according to the constitution that you had. For detailed information for authors, please refer to [1].

## 2. BACKGROUND

This section first explains what the connection means in SOA compared to the web service integration, and then it describes some common test tools of web services including *SoapUI* used in this paper.

### 2.1 Service Connection in SOA

#### 2.1.1 Layering System in SOA

Contemporary SOA is a complex and sophisticated architectural platform that offers significant potential to solve many historic and current IT problems. Three of the primary influences of contemporary SOA are shown in Fig. 1.

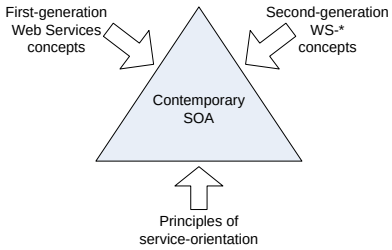



Fig. 1. External influences that form and support contemporary SOA [7]

Contemporary SOA organizes Web services according to the principles of service-orientation; *autonomy*, *reusability*, *enterprise-wide loose coupling*, *extensibility*, *layers of abstraction*, *orchestration*, and so on. Service-orientation principles fully promote black box type abstraction on a service interface level. Fig. 2 shows three primary service layers and three specific service layers of the service interface layer; application service layer, business service layer, and orchestrations service layer.
The application service layer establishes the ground level foundation that exists to express technology-specific functionality. Services that reside within this layer can be referred to simply as application services, $S_A$. Their purpose is to provide reusable functions related to processing data within new or legacy application environment. Some SOA systems can work with only application services. Their maturity level in terms of SOA is said to be low.

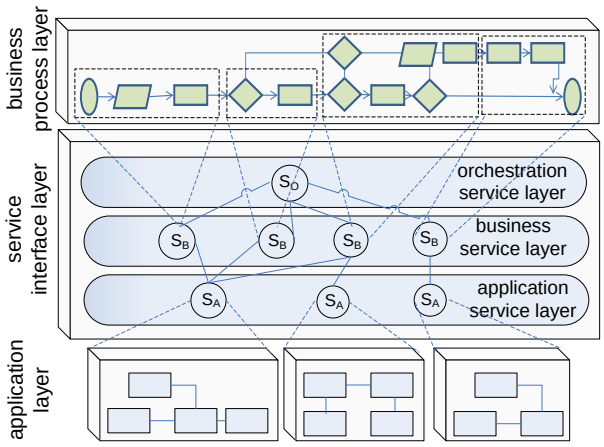



Fig. 2. Layers of SOA architecture [7]

The business service layer introduces services concerned solely with representing business logic, called business services, $S_B$. They act as controllers to compose available application services to execute their business logic. Systems with business services are in the much higher maturity level in terms of SOA.
The orchestration service layer is a parent level of abstraction that alleviates the need for other services to manage interaction details required to ensure that service operations are executed in a specific sequence. Within the orchestration service layer, process services compose other services that provide specific sets of functions, independent of the business rules and scenario-specific logic required to execute a process instance.

#### 2.1.2 Integration vs. Connection in SOA

The services in the same layer have no direct connections, since the *service connections* are set between services in different layers. The fact that web services communicate within the business service layer or the application service layer means they implement the functions supporting similar business tasks or application operations. Therefore, web services that communicate with each other within one layer are strongly recommended to be "integrated" into one SOA service as shown in Fig. 3. It can guarantee that the architecture is loosely-coupled and abstract to business process changes or an application version-up.

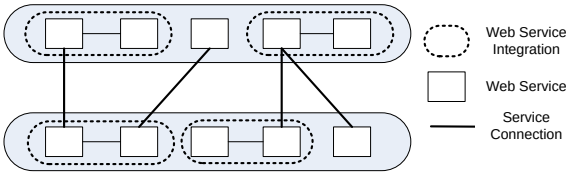



Fig. 3. Web Service Integration vs. Service Connection

Through identifying SOA services, a set of web services deployed in the same layer is recommended to be identified as one SOA service. It is because a service in SOA is autonomous. If two different services are working together in the business service layer, they might support the same or similar task of their business process. If two different services are communicating in the application service layer, it might serve the same or similar operation of applications of the application layer. The relations between them could cause a stronger coupling, which is not proper for SOA. The conventional integration of web services is applied to the integration of web services in the process of identifying SOA services. Therefore, the bold lines of Fig. 3 mean connections between SOA services, and they exist between two different layers.

### 2.1.3 Testing of Web Service Integration

Web service is one of the most common ways to build a SOA. Web service providers use the Web Service Description Language (WSDL) [8] to describe the services they provide and how to invoke them. The service providers then register their services in a public service registry using universal description, discovery, and integration (UDDI) [9]. Application programs discover services in the registry and obtain a URL for the WSDL file that describes the service. Then, the applications can invoke the services using the XML-based simple object access protocol (SOAP) in either asynchronous messaging or remote procedure call (RPC) mode [10,11].

Testing issues on web services have been studied by several researchers. They proposed how to test web services in terms of web service unit testing or web service integration testing. Actually, web service unit testing could be solved by traditional white box testing, since most web services are coded in Java. However, web service integration testing is difficult, because the integration works through exchanging message, not through method calls, which is the traditional approach to integrating two units. Some methods [3,4,5] have been developed with focusing on SOAP messaging. A flow graph could be drawn to show which service sends a request message to which service. In [12,13], the flow graph connects two different services evenly without any hierarchical concepts. It is one of the main differences between the web service integration and the SOA service connection.

### 2.2 Testing Tools of Web services

There are common test tools for web services; such as *SoapUI*[14], *JUint*[15], *TestMaker*[16], *WebInject*[17]. They generate SOAP messages for testing automatically by analyzing WSDL and its schema. *JUnit* is a test solution based on Java technology, and it support Test Driven Development (TDD) practically for Java developers. *TestMaker* is an open source tool for web services. It provides a graphical environment and it supports a script language, *Python*. *TestMaker* makes test cases from WSDL. *WebInject* tests web applications and web services, and it records test results as well as test cases on an XML file.

Table 1. Some test tools of web service

| | ***SoapUI*** | ***Junit*** | ***TestMaker*** | ***WebInject*** |
|---|---|---|---|---|
| **IDE** | Eclipse, NetBens, IntelliJ | Eclipse, NetBens | Eclipse, NetBens | |
| **Test** | **Function Test Load Testing** | **Unit Test Function Test** | **Intelligent Test Functional unit or system Test** | **Function Test Regression Test** |

This paper uses *SoapUI* to show how the test requirements are applied to a test tool, since many web service developers prefer *SoapUI* to the others. It generates SOAP messages of test steps only by including proper WSDL files, and it also shows the replies of the messages on user's windows.

## 3. Test Requirement for Connections

Test requirements for SOA services need to be generated according to the SOA connection principles. Something to show the connection as a diagram would help not to pass over the principles. That's the why *service message flow diagram (SMFD)* is defined in this paper. The diagram is used to generate test requirements for connections.

### 3.1 Service Message Flow Diagram (SMFD)

In the diagram, the request-response mechanism between services is expressed as edges. The service is one of two kinds; business service and application service. A business service is a collection of tasks implementing a business process, and an application service is a collection of operations of the technical implementation. In an SMFD, a node could be a task or an operation depending on the type of service that the task or the operation comes from. SMFD is defined in Definition 1.

**Definition 1.** *Service Message Flow Diagram (SFMD)*
SFMD is an acyclic spanning tree, G(V,E).

· V represents an orchestration service, a business service task, or an application service operation.

· E represents a message flow where a parent requests a child's service.

· $(p,q) \in E$, where $p \in V$ is the parent of $q \in V$.

· V has a name representing which operation of which service the vertex is.

A SMFD is drawn from the architecture shown in Fig. 4 by decomposing a service by a task or an operation and drawing a node for each task or operation. As shown in Fig. 4, a business service is related to a task included in a business process, and an application service is related to an operation. Nodes for tasks should be located in a higher level of the tree than nodes for operations in SMFD. For the sake of convenience, nodes have their own name that represent which operation of which service they are. For example, a node called $S_{A1}3$ is the third operation of an application service named $S_{A1}$.

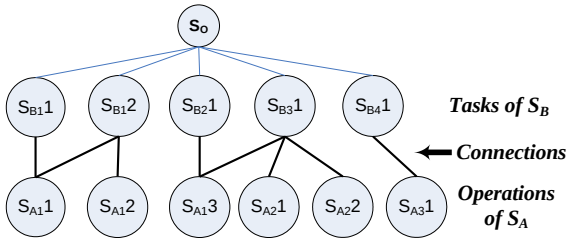


Fig. 4. Task and Operation of a SMFD

## 3.2 Test Requirements

The more complicated the connections are, the more transactions are feasible. A transaction is a set of messages that are exchanged to do a meaningful task of business processes. In SMFD, a transaction is made of edges from task nodes to terminal nodes, and the edges should be covered by the connection testing. The requirements to test connections are defined as a set of paths in a SMFD.

Test requirements need to cover all the connections. A connection is made of message flows between services; not between operations and not between tasks. A task of business services sends a request-message to an operation of application services, and then the operation sends a response-message back to the task. Therefore, the connection in SOA is decided by a pair of a task and an operation. It would be hard to know which message flows should be covered by test data if connections of service units are described. To get the information of a connection, we need something more than relationships of service units. That is the SMFD defined in Section 3.1.

A node in a SMFD means a task or an operation depending on which kind of service the node comes from. An edge in a SMFD connects a task with an operation. All the connections are drawn across different layers in a SMFD. It means that an operation does not send a request or a response to any other operation, nor does a task sends a request or not to any other tasks. SOA does not recommend a connection between operations or a connection between tasks, as the connection could cause a tight coupling of services. An edge in SMFD satisfies these requirements of SOA connections. It is located across the layers, and it connects a task and an operation. Therefore, the edges are what the connection testing covers. They are defined as the test requirements in Definition 2.

**Definition 2.** *Test requirements for service connections (TR)*

The edges in a SMFD represent the qualified connections in SOA. Therefore, connection testing should check if the response returned to the task of an edge is the same as the expected output of the connection.

As explained in Definition 2, an edge shows one transaction where a task requests to an operation and then the operation responds back to the task. The response from an operation is what a testing engineer compares with the expected output. Consequently, the connection that testing should cover is put across the business service layer and the applications service layer, and it links a task to or from an operation as shown in Figure 4. In the SMFD shown in Figure 4, the following set of test requirements is generated. *RR*(x) means the responses of an edge, x, and E(x) means the expected output with an edge, x.

$TR$={$RR(S_{B1}1,S_{A1}1)=E(S_{B1}1,S_{A1}1)$,
$RR(S_{B1}2,S_{A1}1)=E(S_{B1}2,S_{A1}1)$,
$RR(S_{B1}2,S_{A1}2_{)}=E(S_{B1}2,S_{A1}2)$,
$RR(S_{B2}1,S_{A1}3)=E(S_{B2}1,S_{A1}3)$,
$RR(S_{B3}1,S_{A1}3)=E(S_{B3}1,S_{A1}3)$,
$RR(S_{B3}1,S_{A2}1)=E(S_{B3}1,S_{A2}1)$,
$RR(S_{B3}1,S_{A2}2)=E(S_{B3}1,S_{A2}2)$,
$RR(S_{B4}1,S_{A3}1)=E(S_{B4}1,S_{A3}1)$}

In the case of $RR(S_{B1}1,S_{A1}1)=E(S_{B1}1,S_{A1}1)$, $S_{A1}1$ takes a request message for the task $S_{B1}1$, and then $S_{A1}1$ returns a response of the request back to the same task, $S_{B1}1$. The returned response should be the same as its expected output. It is one of the elements of TR. Existing test data selection [3,4,5] based on Web service's SOAP messages could be applied to cover the test requirement, TR, described above.

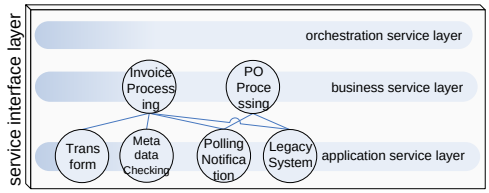
# 4. A Case Study

This section describes two ways to apply the test requirements in order to show the connection test requirement is working practically. First, this paper defines test requirements of two SOAs of two real organizations. Second, it shows the test requirements work on a common test tool of web services. The test tool of web service is countable here, because many SOA developers are still on web services techniques and the web service is still one of the most popular implementation for SOA.

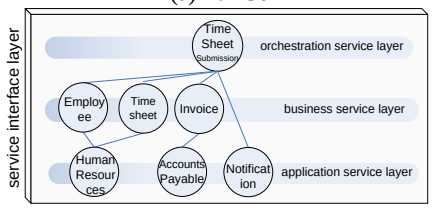
## 4.1 Application to Enterprise Systems

### 4.1.1 Two enterprise systems

The first company is RailCo, a mid-size company with a modest IT staff. The second, Transit Line Systems Inc. (TLS), is a larger corporation with multiple IT departments managing enterprise-level solutions. The names of these companies are aliases, since they do not want to open their real names to the public. The two companies also have a business relationship. RailCo has just upgraded their system in order to perform transactions via B2B solutions with TLS, which has already built B2B solution on a mature SOA architecture. TLS has a higher maturity level than RailCo in terms of the SOA maturity level.




(a) *RailCo*




(b) *TLS*

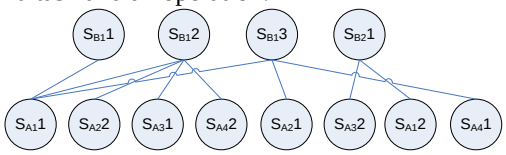
Fig. 5. Service deployment on layers [2]

Fig. 5 shows the architectures of RailCo and TLS. The architecture of RailCo has no orchestration service layer because the company did not have budget for building a middleware. Instead, business services control application services to follow its business process. In this sense, RailCo's business services are task-centric business services, while TLS's are entity-centric business services. They might be refined as orchestration services in the future.

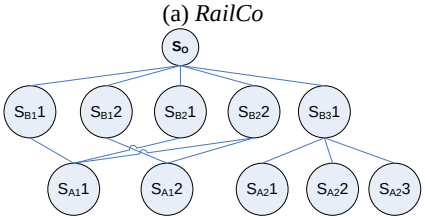
### 4.1.2 SMFDs

Before drawing the organizations' SMFDs of Fig. 5, a service should be decomposed by a task or an operation, and connections should be set between tasks and operations. We describe services, tasks, operations, and moreover their connections in a table.

From the connection tables, nodes and edges are drawn as in Fig. 6 (a) and (b). Two things look weird in Fig. 6. One is that RailCo's SMFD shown in Fig. 6 (a) has no root. It is because RailCo has not built any process service in the orchestration service layer due to budget constraints as mentioned in Section 4.1.1. The other thing to look weird is that TLS's SMFD shown in Fig. 6 (b) does not draw the operation of $S_{A3}$. The request message flow of a process service, SO, and the operation, $S_{A3}1$ is not shown in SMFD. It is because $S_{A3}$'s operation is requested by a process service of the orchestration service layer not by a business service. As explained in Fig. 4, the connection is between a task and an operation.




(a) *RailCo*




(b) *TLS*

Fig. 6. SMFDs of TLS and RailCo

### 4.1.3 Test Requirement

(1) RailCo

There are ten connections in the SMFD of RailCo as shown in Fig. 6 (a). It means that the connection testing needs to cover all the 10 connections of tasks and operations. According to Definition 2, the TR for RailCo is as follows.

$TR$={$RR(S_{B1}1,S_{A1}1)=E(S_{B1}1,S_{A1}1)$,
$RR(S_{B1}2,S_{A1}1)=E(S_{B1}2,S_{A1}1)$,
$RR(S_{B1}2,S_{A2}2)=E(S_{B1}2,S_{A2}2)$,
$RR(S_{B1}2,S_{A3}1)=E(S_{B1}2,S_{A3}1)$,
$RR(S_{B1}2,S_{A4}2)=E(S_{B1}2,S_{A4}2)$,
$RR(S_{B1}3,S_{A1}1)=E(S_{B1}3,S_{A1}1)$,
$RR(S_{B1}3,S_{A2}1)=E(S_{B1}3,S_{A2}1)$,
$RR(S_{B1}3,S_{A4}1)=E(S_{B1}3,S_{A4}1)$,
$RR(S_{B2}1,S_{A1}2)=E(S_{B2}1,S_{A1}2)$,
$RR(S_{B2}1,S_{A3}2)=E(S_{B2}1,S_{A3}2)$}

For instance, in case of $RR(S_{B1}3,S_{A1}1)=E(S_{B1}3,S_{A1}1)$, $S_{B1}3$ is a task to 'send an electronic invoice to a service' as described in Table 2. It sends a request message to $S_{A1}1$, which 'exports document to network folder' as seen in Table 2. This connection is notated as the edge, ($S_{B1}3$, $S_{A1}1$), in Fig. 6 (a). $RR(S_{B1}3,S_{A1}1)=E(S_{B1}3,S_{A1}1)$ in TR means that a test data should check if the response back to $S_{B1}3$ from $S_{A1}1$ is equal to its expected output. $S_{B1}3$ begins this transaction by sending a request message to $S_{A1}1$.

(2) TLS

TR of TLS consists of eight elements, which are edges between nodes of a task of a business service and nodes of an operation of an application service. The edges from $S_O$

are not counted as connections of services, since the process service in the orchestration service layer work as a middleware. It controls or manages the process, but it does not request other service's functions for executing SOA systems. The RailCo system works even though it has no process services in the architecture. Therefore, the connections that the testing should cover in Fig. 6 (b) are limited to those of tasks and operations. The TR for TLS is generated as follows.

$TR$={$RR(S_{B1}1,S_{A1}1)=E(S_{B1}1,S_{A1}1)$,
$RR(S_{B1}2,S_{A1}2)=E(S_{B1}2,S_{A1}2)$,
$RR(S_{B2}1,S_{A1}1)=E(S_{B2}1,S_{A1}1)$,
$RR(S_{B2}2,S_{A1}1)=E(S_{B2}2,S_{A1}1)$,
$RR(S_{B2}2,S_{A1}2)=E(S_{B2}2,S_{A1}2)$,
$RR(S_{B3}1,S_{A2}1)=E(S_{B3}1,S_{A2}1)$,
$RR(S_{B3}1,S_{A2}2)=E(S_{B3}1,S_{A2}2)$,
$RR(S_{B3}1,S_{A2}3)=E(S_{B3}1,S_{A2}3)$}

In case of $RR(S_{B2}2,S_{A1}2)=E(S_{B2}2,S_{A1}2)$, $S_{B2}2$ is a task to 'reject timesheet' as described in Table 3. It sends a request message to SA12, which 'comments' as seen in Table 3. This connection is notated as the edge, ($S_{B2}2$, $S_{A1}2$), in Figure 6 (b). $RR(S_{B2}2,S_{A1}2)=E(S_{B2}2,S_{A1}2)$ in TR means that a test should check if the response back to $S_{B2}2$ from $S_{A1}2$ is equal to its expected output. $S_{B2}2$ begins this transaction by sending a request message to $S_{A1}2$.

### 4.2 Application to a Testing Tool, *SoapUI*

A connection is made up of a business service and an application service. The business service in one connection requests a specific operation of an application service to work its part of the business process. This service connection is working through an orchestration of services as an intermediate. Using an intermediate enhances SOA's loosely coupling of services and it guarantees that every service has no direct coupling each other in SOA.

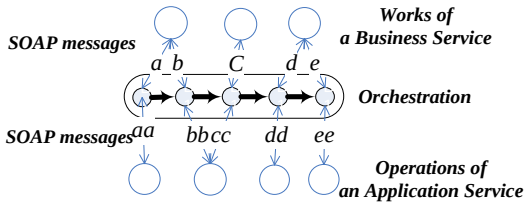



Fig. 7 Connection in Orchestration

Fig. 7 shows how an orchestration works with business services and application services. For one work of a business process, an orchestration first sends a request message to its appropriate business service implementing behaviors of the work, and then it receives the response from the service. The response is supposed to be transferred to its matching application service having an operation to implement the business work in IT level. For example, a web page getting information could be a business service and an application handling the information taken from the web page could be its matching application service.

As shown in Fig. 7, a message, *a*, is sent to a business service by an orchestration, and then the following message, *aa*, is sent to its matching application service. A pair of these two request messages describes one connection of the services. Therefore, the test steps for testing connection should keep the sequence of two messages as an atomic. A pair of a message to business services and a message of their matching applications is denoted as *(a,aa)* for the convenience's sake. To test the orchestration of Fig. 7, the connection testing runs the test steps including *(c,cc)*, *(d,dd)*, *(e,ee)* as well as *(a,aa)*,*(b,bb)*.

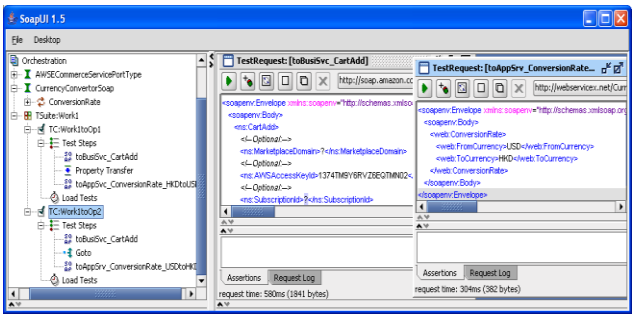



Fig. 8. Test Steps of Orchestration

Fig. 8 shows that a testing tool, *SoapUI*, set the test suite for the orchestration. In case that SOA developers use *SoapUI* as a testing tool, each test steps for testing an orchestration need to include an atomic sequence of a request messages as described before. As shown in Fig. 8's test steps of the test cases, "*TC:Work1toOp1*" and "*TC:Work1toOp2*", "*toBusiSvc_CartAdd*" is a request message to *CartAdd* defined in a business service, *AWSEcommerceServicePortType*[18], and right after that, "*toAppSvc_ConversinRate_HKDtoUSD*" is a request message to ConversionRate defined in an application service, *CurrencyConvertorSoap*[19], with the value, *HKD* and *USD*, in the message as shown in Figure 3. "*toBusiSvc_CartAdd*" could be *a* of Fig. 7, and "*toAppSvc_ConversinRate_HKDtoUSD*" could be *aa* of Fig. 7.

## 5. Conclusions and Future Work

This paper developed the test requirements for SOA service connection, which was defined separately from service integrations. As examples, it also applied the test requirements to two SOA cases of real companies, TLS and RailCo. In addition, it built test steps supporting the test requirement in a common test tool of web services, because many SOA developers still stay on web service

technology, and one of SOA implementation techniques is an implementation of web services.

Currently, many SOA implementations put all their services on a hybrid service layer instead of a business service layer and an application service layer separately, where it means that they don't decide if a service supports a business process or its application. Through improving the system to a more mature SOA system, it distinguishes services for a business process from services for its applications. That makes the system have two separate service layers; business service layer and application service layer. This paper focused on the system with these separate layers.

A service connection is a new important spot of testing in SOA as talked in some SOA conferences. Compared to the conventional integration, this paper first clearly defined the service connection based on the SOA layering system. The SOA layering system is strongly recommended for keeping the loosely-coupled between a business process and its applications. If services are deployed on a single layer, a business process and its applications would be tight coupled. This paper caught service connections by defining a graph, SMFD. A node in SMFD is a task of business services or an operation of application services depending on which kind of service the node comes from, and an edge is a connection itself between them. Edges in SMFD are transformed into test requirements, TR. We are improving SMFD to cover B2B connections. Actually, TLS and RailCo aimed to make a connection between their systems. It was the motivation that RailCo reorganized its non-SOA system to SOA architecture. In Fig. 6, $S_{B1}2$ of RailCo can request SA21 of TLS as a B2B connection. This connection will be covered in our next version of SMFD.

To test service connections completely, a test data selection criteria is needed. We are working on selecting effective test data with satisfying the test requirement this paper proposed. The test data can be used to fill in the blanks of SOAP request messages of *SoapUI*'s test steps. With the test data selection criteria, some experimental studies will evaluate the effectiveness of test data selected by the criteria.

## References


[1] Eric A. Mark, A Planning and Implementation Guide for Business and Technology, John Wiley and Sons, 2006.

[2] HP Software Universe 2007, http://h30350.www3.hp.com/conference/index.jsp, 2007

[3] Hai Huang, Wek-Tek Tsai, Raymond Paul, and Yinong Chen, "Automated Model Checking and Testing for Composite Web Services," Proceeding on IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2005

[4] Xianoying Bai and Wenli Dong, "WSDL-Based Automatic Test Case Generation for Web Service Testing," Proceeding on IEEE International Workshop on Service-Oriented System Engineering, 2005

[5] Jeff Offutt and Wuzhi Xu, "Generating Test Cases for Web Services Using Data Perturbation," Proceeding on Workshop on Testing, Analysis and Verification of Web Services, pp.41-50, July 2004

[6] Robert Peterson, "Get started with WebSphere Integration Developer," IBM WebSphere Developer Technical Journal, 2005.12.07

[7] Thomas Erl, Service-Oriented Architecture – Concepts, Prentice Hall, 2005

[8] Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, and Sanjiva Weerawarana, "Web Services Description Language (WSDL) Version 1.2," W3C Recommendation, 2002, www.w3.org/TR/wsdl12

[9] "Universal Description, Discovery, and Integration," OASIS standard, 2002

[10] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana, "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," IEEE Internet Computing, Vol.6, No.2, Mar./Apr. 2002, pp.86-93

[11] D. Box, D. Ehunbuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," W3C Note 08, May 2000

[12] Daniel A. Menascé, "Composing Web Services: A QoS View," IEEE Internet Computing, pp.88-90, Nov./Dec. 2004

[13] Daniel A. Menascé, "QoS Issues in Web Services," IEEE Internet Computing, pp.72-75,Nov./Dec. 2002

[14] SoapUI – Web Service Testing, http://www.soapui.org/

[15] JUnit - Test Driven Development , http://www.junit.org

[16] PushToTest TESTMAKER, http://www.pushtotest.com

[17] webInject – web/HTTP Test Tool, http://www.webinject.org

[18] Amazon Web Service Ecommerce Service, http://webservices.amazon.com/AWSECommerseService/AWSECommerseService.wsdl

[19] Currency Converter Web Service, http://www.webservicex.net/CurrencyConvertor.asmx?WSDL


**Hoijin Yoon** received the B.S. and M.S. degrees in Computer Science and Engineering from Ewha Womans University in 1993 and 1998, respectively. She also received her ph.D with the dissertation about software component testing from Ewha. After the degree, she stayed in Georgia Institute of Technology as a visiting scholar and then worked at Ewha Womans University. She has been teaching at Hyupsung University as a full-time lecturer since 2007, She is interested in Software Testing, Service Oriented Architecture, and Testing in Cloud computing.