Automating the DEVS Modeling and Simulation Interface to Web Services

Chungman Seo
Bernard P. Zeigler
Arizona Center for Integrative Modeling and Simulation
The University of Arizona
Tucson, AZ
cseo, zeigler@ece.arizona.edu

Keywords: DEVS, Web Services, WSDL, Dynamic Web Service Invocation, Web Service Execution Language

Abstract

Web service technology is used to augment software reusability and composability providing XML-based message passing and operating system platform neutralization. Business Process Execution Language (BPEL) is commonly used to integrate web services called business processes, and implementations of BPEL vary according to venders. BPEL components have simple behavior. In this paper, we propose automatic generated DEVS interface to web services to apply a formal method to orchestrating web services to provide dynamic testing environment of business processes with DEVS modeling and simulation and to enhance the reusability of business processes and interoperability of instances of BPEL. We will show an environment of DEVS interface for Web Services to integrate and execute web services, and describe its components such as integration of web services, a XML document describing the integration, and creation and execution of DEVS models with a mechanism of dynamic invocation of web services.

1. INTRODUCTION

Service Oriented Architecture (SOA) is one of the software architecture concepts. It enables the reusability and composability of software to increase with web services based on a combination of the Web, XML, Simple Object Access Protocol (SOAP), and Web Service Description Language (WSDL) [1-4]. The web services define their message structures, operations, and locations in the WSDL, and communicate to a service requester with SOAP messages which provides platform independence and neutral messages.

Web services can be composed to create other business process with Business Process Execution Language (BPEL) which integrates business processes distributed on the web. BPEL provides activities and structures to invoke and control the web services like computer languages e.g. JAVA, C++. BPEL is written by a XML based document including web service location, message types, and elements of BPEL (e.g. activities, structures, handlers). To effectively share the intention of a designer of business processes with an implementer, Business Process Model Notation (BPMN) [5-7] is developed where some graphical notations make design of a business process model clearer than explanation of it.

While BPEL and BPMN are used to easily create business processes, it is not easy to test the business processes with dynamic scenarios and to interoperate between BPEL models in the different vendors because commercial products of BPEL, such as ActiveBPEL, ParasoftBPEL, Oracle, and IBM [8-11], use their different methods to implement BPEL engines. BPEL model can be tested using WSUnit which generates consistently and repeatedly web service responses [12]. WSUnit just invokes web services with simple, fixed data without dynamic manners. To overcome the problems, formal modeling methodology should be applied to integrate web services.

Discrete Event System Specification (DEVS) is suitable for solving a dynamic testing problem and an interoperable problem between BPEL models implemented by different providers. There are two ways to convert web services to DEVS models. One method is to use BPEL models described in XML where BPEL to DEVS converter using XML parsers for WSDL and BPEL maps BPEL tags to atomic DEVS formalism. The mapping mechanism of all components of BPEL is required to make a generic BPEL/DEVS converting engine as described in [13]. But [13] did not include a method of invocation of web services. The other is to create an environment of DEVS interface for Web Service (DEVSI4WS) using the WSDL. This approach provides a method of dynamic invocation of web services.

In this paper, DEVSI4WS is proposed to integrate and execute business processes. It consists of integration of web services, a XML document containing information of web service, creation of DEVS models, and execution of DEVS models. The DEVS models embed the dynamic invocation mechanism of web services. To prove this concept, we demonstrate integration of simple web services using the DEVSI4WS system.

In the rest of the paper, the background of DEVS and SOA is discussed in the section 2. The section 3 addresses an overall architecture of web services integration and execution, and its functions such as message inclusion, dynamic invocation, and conversion of web services to DEVS models. The section 4 explains web service integration environment. The example of integration of web service is presented with a simple web service in the section 5. The paper's summary and future works are in the section 6.

2. BACKGROUND

2.1. Discrete Event System Specification (DEVS)

The Discrete Event System Specification (DEVS) is a formalism describing entities and behaviors of a system [14]. There are two kinds of models in DEVS which are atomic and coupled models. An atomic model depicts a system as a set of input/output events and internal states along with behavior functions regarding event consumption/production and internal state transitions. A coupled model consists of a set of atomic models, coupling information among the atomic models, and input/output ports.

The Atomic model can be illustrated as a black box having a set of inputs(X) and a set of outputs(Y), or a white box specifying a set of states(S) with some operation functions (i.e., external transition function (δ_{ext}), internal transition function (δ_{int}), output function (λ), and time advance function (ta())) to describe the dynamic behavior of the model. The external transition function (δ_{ext}) carries the input and changes the system states. The internal transition function (δ_{int}) changes internal variables from the previous state to the next when the time advance is expired and no events have occurred since the last transition. The output function (λ) generates an output event in the current state. The time advance (ta()) function determines the time to stay in the state after generating an output event. The Atomic model is specified as follows:

$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda , ta >$

A coupled model is the major class which embodies the hierarchical model composition constructs of the DEVS formalism. A coupled model is made up of component models, and the coupling relations which establish the desired communication links. A coupled model illustrates how to connect several component models together to form a new model. Two significant activities involved in coupled models are specifying its component models and defining the couplings which create the desired communication networks

2.2. Service Oriented Architecture (SOA) and Web Service

SOA is a methodology with which a new application is created through integrating existing and independent business processes which are distributed over the networks. The business processes are called modules or services which communicate with each other, passing a message through the networks. This design concept requires interoperability between heterogeneous systems and languages, and orchestration of services to meet the purpose of the creator.

One of the implementation of SOA concept [1] is web service which is a software system for communicating between a client and a server over a network with XML messages called Simple Object Access Protocol (SOAP) [15]. The web service makes the request of machine-to-machine or application-to-application communication possible with neutral message passing even though each machine or application is not same domain. Web service realizes interoperability among different applications providing a standard means of communication and platform independence.

Web services technologies architecture [2] is based on exchanging messages, describing web services, and publishing and discovering web service descriptions. The messages are exchanged by SOAP messages conveyed by internet protocol. Web services are described by Web Services Description Language (WSDL) [3] which is XML based language providing required information, such as message types, signatures of services, and a location of services, for clients to consume the services. Publishing and discovering web service descriptions is managed by Universal Description Discover and Integration (UDDI) which is a platform-independent and XML style registry. In other words, three roles are classified in the architecture that is, a service provider, a service discovery agency (UDDI), and a service requestor. The interaction of the roles involves publishing, finding, and binding operations. A service provider defines a service description for a web service and publishes it to a service discovery agency. This operation is publishing operation between the service provider and the service discovery agency. A service requestor uses a finding operation to retrieve a service description locally or from a discovery agency and uses the service description to bind it with a service provider and invoke or interact with the web service implementation. Figure 1 illustrates the basic Web services architecture describing three roles and operations with WSDL and SOAP.

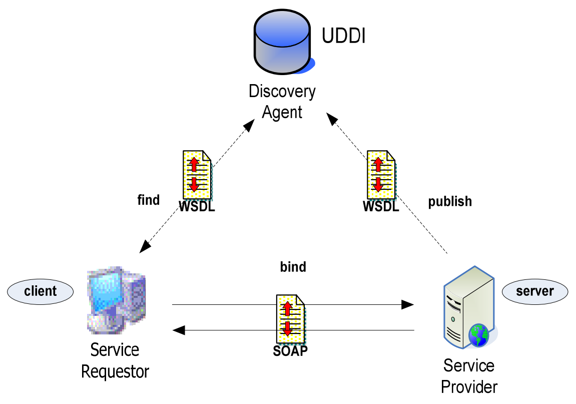


Figure 1 Web Services Architecture

Whereas a web service is an interface described by a service description, its implementation is the service which is a software module provided by the service provider (server) on the network accessible environment. It is invoked by or interacts with a service requestor (client).
Web services are invoked by many ways but the common use of web services is categorized to three methods such as Remote Procedure Call (RPC), Service Oriented Architecture (SOA) [16], and Representational State Transfer (REST). RPC Web services was the first web services approach which had a distributed function call interface described in the WSDL operation. Though it is widely used and upheld, it does not support loosely coupled concept for reasons of mapping services directly to language-specific functions calls. Other web service is an implementation of Service-Oriented Architecture (SOA) concepts, which means a message is important unit of communication regarded as "message-oriented" services. This approach supports a loose coupling concept focusing on the contents of WSDL. REST Web services focuses on the existence of resources rather than messages or operations. It considers WSDL as a description of SOAP messaging over HTTP, or is implemented as an abstraction on top of SOAP.

3. OVERALL ARCHITECTURE OF WEB SERVICES INTEGRATION AND EXECUTION

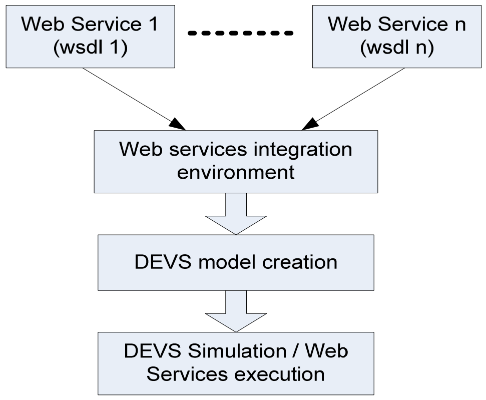

Web Service 1 (wsdl 1)
Web Service n (wsdl n)
Web services integration environment
DEVS model creation
DEVS Simulation / Web Services execution

Figure 2. Overall architecture of Web services Integration and Execution

Figure 2 shows an overall architecture for web services integration and execution using DEVS modeling and simulation. This system consists of three major parts which are Web Services Integration Environment (WSIE), DEVS model creation, and Web services execution through the DEVS simulation.

WSIE is a place to gather required elements to describe the integration of web services as a DEVS coupled model with WSDLs that contain information of web services such as input and output message types, service names, and service locations. The elements represent web services which are converted to DEVS atomic models, coupling information among DEVS atomic models, and input/output ports information. The coupling information is based on service names and message types from services that have input messages and output messages. To connect a model to other models, common information between an output message of a sender and an input message of a receiver should be found. This commonality is checked by a message inclusion method. The generation of an input port and an output port in WSIE increases reusability of web services through integrating with other DEVS models using the ports.

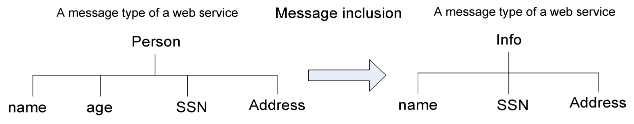
WSIE generates an XML based description language containing the information of mapping web services to DEVS atomic models and a DEVS coupled model. A schema for the description language is defined to check validity of the XML document.

DEVS model creation generates a DEVS coupled model with the information from WSIE and each DEVS atomic models encapsulating each operation of web services according to mapping the information of a web service to DEVS formalism explained in detail later. To extract the information of the XML document, Java Architecture for XML Binding (JAXB) is applied. A schema is converted to the set of Java classes to make marshaling XML to Java codes and unmarshaling Java codes to XML easy by JAXB [17].

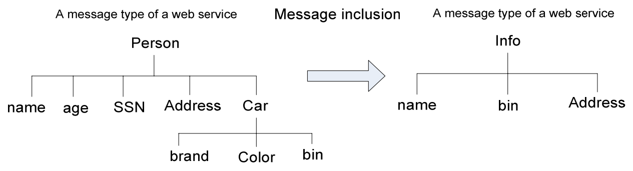
After the creation of all DEVS models including coupled and atomic DEVS models, all DEVS models are ready to be simulated by a DEVS simulation environment providing several options to execute DEVS models in which there is a mechanism that invokes a web service dynamically.

3.1. Message Inclusion

WSDL displays the data types consumed in the client who invokes the web service. The data types are expressed by a schema describing structures of each data type. When web services are integrated like Business Process Execution Language (BPEL), message conversion is required to connect from a web service to other web service. In WSIE, message inclusion method is used, which checks the possibility of connection between output message and input message.


A message type of a web service
Message inclusion
A message type of a web service
Person
name
age
SSN
Address
Info
name
SSN
Address

a) Simple data type



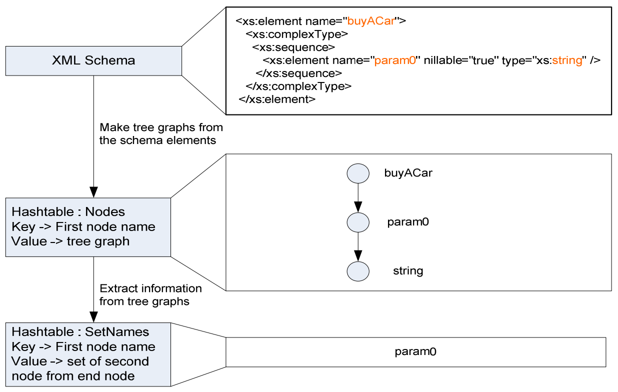
b) Complex data type

Figure 3. Examples of message inclusion

Figure 3 represents examples of message inclusion. In case of simple data type, *Person* type can include *Info* type because all elements of *Info* type can be extracted in all elements of *Person* type. In the complex data type, the *Person* type has a type of *Car* and other primitive types such as String, int, double, and so on. The *Info* type requires one element from the *Car* type. To search elements of *Info* type in the *Person* type, all elements of the end nodes in the *Info* type is selected and compared with all elements of the end nodes in the *Person* type.

To invoke two web services as a sender and a receiver, the result of invocation of the sender should include the request message of the receiver. Once satisfying the message inclusion relation, two web services can be invoked sequentially.

Figure 4 depicts an example of conversion of the data type of schema to tree data structure to save ordered information of each data type. The *buyACar* type has one element called parameter0 with string type as seen in the figure 4. The *buyACar* type is represented to a tree structure which has three nodes. To store the tree structure, the hashtable called Nodes which consists of a key and a value is used. The key is the root node name and the value contains the tree structure. From the tree structure, names of elements of a type are extracted and saved on a set as a value in the hashtable called SetNames with the name of the root node as a key.

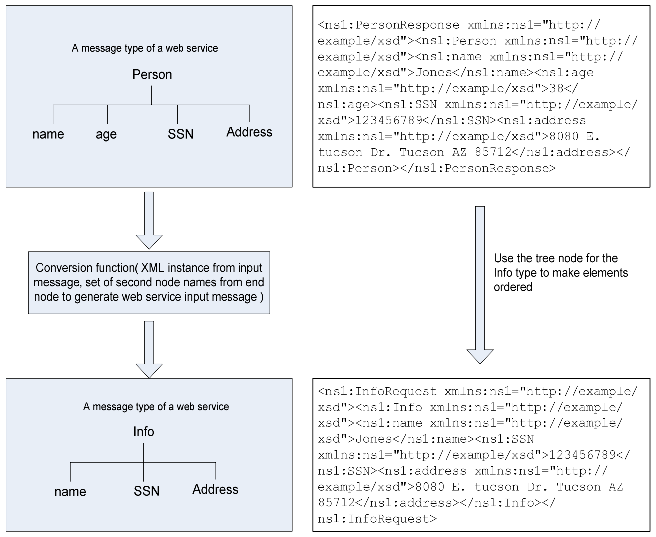


```
<xs:element name="buyACar">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="param0" nillable="true" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 4. The making of tree structures from the schema

In case that a message type has an element represented to a complex type, tree structures need to be changed to construct whole tree structures. This operation is called node compression For example, in the complex data type of figure 3, the *Person* type has a Car type which is complex data type including three primitive types. The tree structure of the *Person* type should include the tree structure of the *Car* type to express the *Person* message.

Nodes hashtable is used to create an instance of the message. Each node from a data type is shown as a tag name in the XML message except the end node which is represented to a value. SetNames hashtable is employed to decide message inclusion.



```
<ns1:PersonResponse xmlns:ns1="http://
example/xsd"><ns1:Person xmlns:ns1="http://
example/xsd"><ns1:name xmlns:ns1="http://
example/xsd">Jones</ns1:name><ns1:age
xmlns:ns1="http://example/xsd">38</
ns1:age><ns1:SSN xmlns:ns1="http://example/
xsd">123456789</ns1:SSN><ns1:address
xmlns:ns1="http://example/xsd">8080 E.
tucson Dr. Tucson AZ 85712</ns1:address></
ns1:Person></ns1:PersonResponse>
```

```
<ns1:InfoRequest xmlns:ns1="http://example/
xsd"><ns1:Info xmlns:ns1="http://example/
xsd"><ns1:name xmlns:ns1="http://example/
xsd">Jones</ns1:name><ns1:SSN
xmlns:ns1="http://example/xsd">123456789</
ns1:SSN><ns1:address xmlns:ns1="http://
example/xsd">8080 E. tucson Dr. Tucson AZ
85712</ns1:address></ns1:Info></
ns1:InfoRequest>
```

A message type of a web service
Person
name
age
SSN
Address
Conversion function(XML instance from input message, set of second node names from end node to generate web service input message)
Use the tree node for the Info type to make elements ordered
A message type of a web service
Info
name
SSN
Address

Figure 5. A XML instance using the conversion function

The example of generation of a message is represented in figure 5. The *Person* message should be converted to the *Info* message to be used as a request message when a web service is invoked. To support the construction of the request message, the conversion function is used and consists of two arguments which are XML instance from input message and a set of names of second node from the end node. The right side of figure 5 displays an example of a XML instance of a request message used in the invocation of a web service.

3.2. Mapping a Web Service to a DEVS model

A DEVS model can express system behaviors which are comprised of their states, input messages, output messages, internal and external event handler, and time. Web service has a simple behavior in the point of view of event system. An argument of an operation in the web service is considered as an external event, and a return value of an operation is regarded as an output message. An operation is mapped to an output function.

Figure 6 depicts mapping an operation of a web service to a DEVS atomic model which has an input port as an argument message type, an output port as a return message type, information to invoke a dynamic invocation of a web service such as WSDL, and two states which are active and passive. The DEVS atomic model controls an invocation of an operation through the DEVS simulation protocol. When the DEVS model has an input message, external event handler produces a request message for an operation of web service from the input message and changes a state of the model. When an internal event is triggered, the output function of the model which possesses a dynamic invocation of a web service is called. The output function gets the response of the web service and generates an output message.

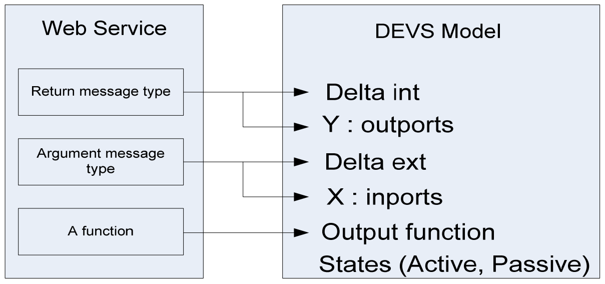


Figure 6. The relation of Web service to DEVS model

The dynamic invocation of a web service needs some information from the WSDL. WSDL describes types tag that are sent and received during the invocation, message tag that includes one type, port type tag that describe the forms of operations, binding tag that indicates a communication method, and service tag that displays the location of the service. Figure 7 describes which information is needed when a web service is invoked dynamically. WSDL-based dynamic invocation function requires five arguments which are a name of WSDL document, a name of an operation, a service location, target name space, an argument of an operation which is a request message. As seen in figure7, target name space, the names of operations, and the service location can be found in the WSDL. To make a request SOAP message, information of types and message on WSDL is used. The dynamic invocation of a web service is implemented with AXIS2 API which provides Java codes to create web services and to invoke web services [18].

The dynamic invocation web service client function returns a response of the web service. The client function consists of an operation client, a request message, and execution of the operation client. Initially a SOAP message is returned into the client function, but it filters the SOAP message to get the body context which is a response message. The response message is a XML document which is handled in DEVS simulation level.

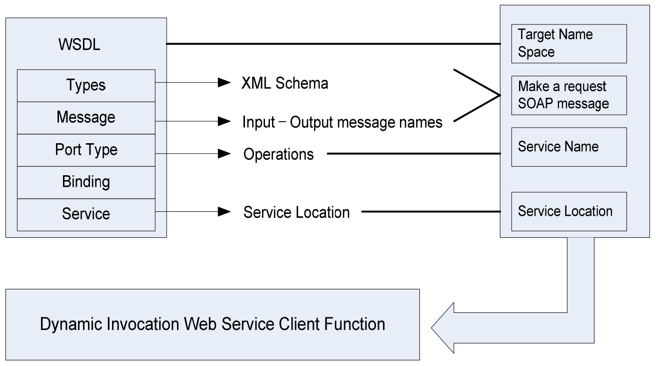


Figure 7. WSDL-based dynamic invocation of a web service with AXIS2

A DEVS message is a set of content classes which consists of a port name and an entity class defined in the DEVSJAVA API provided by Arizona Center for Integrative Modeling & Simulation (ACIMS) [19]. In this system, a class that a user defines inherits the entity class and has two variables, name and contents. The contents variable contains a XML document created by a return value of invocation of the web service.

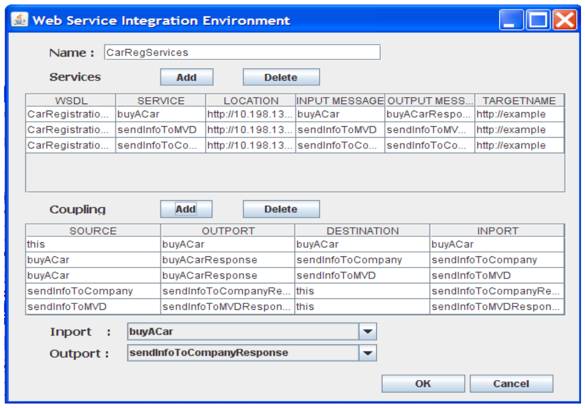
4. WEB SERVICES INTEGRATION ENVIRONMENT

A user can select the information to invoke web services, link between two web services based on message inclusion method, create a XML document including information of operations of web services and coupling information between them, generate DEVS models using the XML document, compile DEVS models, and execute the simulation with WSIE.

Figure 8 represents a user interface to integrate web services. The GUI has five categories which are name, services, coupling, inport, and outport. The name field is utilized as the name of the DEVS coupled model and the XML document. The services field is used to add and delete the information of an operation of a web service with WSDL. Add button by services field makes a GUI for information of a web service displayed. The GUI has some functions to open a specific folder which has WSDL documents, extract information needed to invoke an operation dynamically from a selected WSDL, and write the information to the table under the services field. A row of the table is converted to a DEVS atomic model with columns' information. The coupling field has two buttons and a table to display coupling information. The Add button makes a GUI for coupling shown. The GUI has four combo boxes and labels to select ports and services. If a service is selected as a source, the possible input ports are shown in the combo box by the output message label. Depending on selecting the destination, input message types corresponding with the output message is displayed into the combo box.

The final coupling information is written in the table below the coupling field. The inport field has a combo box containing input ports of the coupled model. The outport field is same as the inport except displaying output ports.

After finishing setting all values using WSIE, OK button on the GUI makes a XML document created in accordance with a schema called devs4ws.xsd. The XML document contains all information from the GUI of the WSIE. The names of each field and the first row of services table and coupling table are represented to tag names in the XML document. The reason making the XML document is for increasing reusability of DEVS models containing web services. Because the XML document expresses a DEVS coupled model, the DEVS coupled model can be used as a component in other coupled model.



Name : CarRegServices

Services

WSDL	SERVICE	LOCATION	INPUT MESSAGE	OUTPUT MESS...	TARGETNAME
CarRegistratio...	buyACar	http://10.198.13...	buyACar	buyACarRespo...	http://example
CarRegistratio...	sendInfoToMVD	http://10.198.13...	sendInfoToMVD	sendInfoToMV...	http://example
CarRegistratio...	sendInfoToCo...	http://10.198.13...	sendInfoToCo...	sendInfoToCo...	http://example

Coupling

SOURCE	OUTPORT	DESTINATION	INPORT
this	buyACar	buyACar	buyACar
buyACar	buyACarResponse	sendInfoToCompany	sendInfoToCompany
buyACar	buyACarResponse	sendInfoToMVD	sendInfoToMVD
sendInfoToCompany	sendInfoToCompanyRe...	this	sendInfoToCompanyRe...
sendInfoToMVD	sendInfoToMVDRespon...	this	sendInfoToMVDRespon...

Inport : buyACar

Outport : sendInfoToCompanyResponse

Figure 8. Web Service Integration Environment

DEVS models are created using the XML document. Java Architecture for XML Binding (JAXB) API is applied to handle the XML document. Once Java codes for the devs4ws schema are obtained by JAXB API, conversion of a Java instance to the XML document or XML to Java makes handling the XML instance easy. So, accessing to a value of a tag is obtained as data type of Java. There are two templates to generate DEVS atomic and coupled model. The template for an atomic model uses the elements of services tag. The other requires coupling tag, in/outport tags, and variables of atomic models because the coupled model should include atomic models.

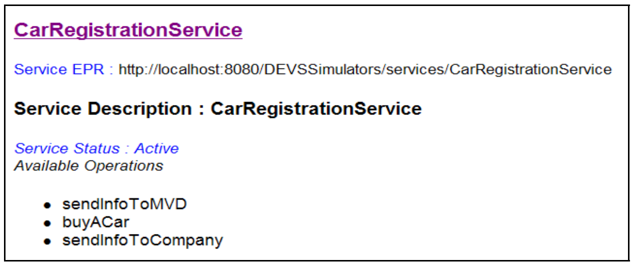
The DEVS models are ready to be simulated after compilation of all DEVS models. The DEVS models are viewed by a simView GUI included in DEVSJAVA API. The simView displays atomic models and coupling between them, and controls simulation of DEVS models with step, run, and restart buttons. Another way to simulate the DEVS models uses input message injection GUI which displays a table having name, type, and value field of an input message. The value field is not determined, so a user should fill the value field to make a request message. After that, pushing the inject button makes simulation run. A user can change the values of the input message through the input message injection GUI.

5. EXAMPLE OF INTEGRATION OF WEB SERVICES

To demonstrate how a DEVSI4WS system works, a web service called CarRegistrationService is created using AXIS2 middleware and apache web server. The web service has three operations as seen in the figure 9. The buyACar operation has an argument and a return type of InfoByCarDealer consisting of nine variables. The sendInfoToMVD and sendInfoToCompany operations have complex data types called an InfoByMVD and an InfoByCompany as an argument and a simple type as a return type.

```
public interface CarRegistrationService {
      public InfoByCarDealer buyACar(String name);
      public String sendInfoToMVD(InfoByMVD info);
      public String sendInfoToCompany(InfoByCompany info);
}
```

Figure 9. The operations of CarRegistrationService



CarRegistrationService

Service EPR : http://localhost:8080/DEVSSimulators/services/CarRegistrationService

Service Description : CarRegistrationService

Service Status : Active
Available Operations

- sendInfoToMVD
- buyACar
- sendInfoToCompany

Figure 10. The web service of CarRegistrationService

Figure 10 displays the actual web service in the Microsoft explorer. The web page contains an address and description of web service and names of operations. It displays the WSDL for CarRegistrationService if clicking the name of web service on the web page.

The scenario of using the above web service is that the information of a client buying a car in the car dealer shop and the car is sent to the Motor Vehicle Division (MVD) for registering an owner and a vehicle, and the company of the sold car for managing their stock. The buyACar operation gives a client the information of a buyer and a vehicle when the client provides the name of the buyer. The sendInfoToMVD requires an InfoByMVD data type and returns a string type. If the buyACar and sendInfoToMVD are connected, the InfoByCarDealer data type should include the InfoByMVD data type. The DEVSI4WS system checks their relation during coupling between them using a message inclusion method. Similarly, the buyACar and sendInfoToCompany is connected.

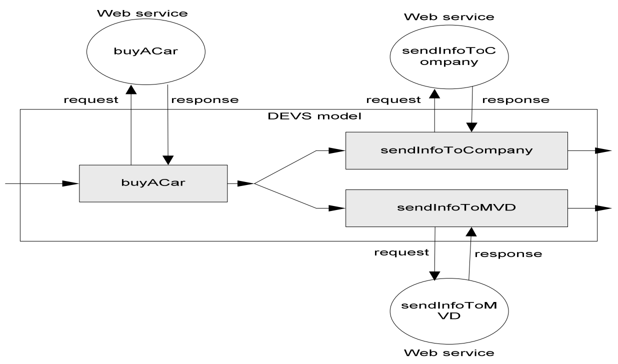


Figure 11. The relation between DEVS atomic models and operations

Figure 11 shows the relation of DEVS atomic models and operations of web service. A DEVS atomic model called a buyACar has a connection to a buyACar operation. Input message of a DEVS coupled model is injected to the buyACar model which converts the input message to a request message, invokes the buyACar operation with the request message, and get a response message. The buyACar model sends the response message to a sendInfoToCompany and sendInfoToMVD atomic models using a DEVS message. The two atomic models use the message conversion function to make request messages for each operation.

Figure 12 shows the view of the DEVS models created by the WSIE displayed in figure 8. The three atomic models are named after each operation name in the WSDL. The input ports and output ports of atomic models are named after the names of arguments and returns of operations. The coupled model can have input ports and output ports if the ports are selected in the WSIE.

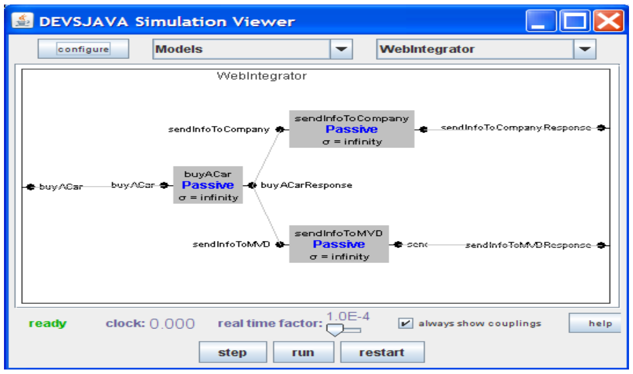


Figure 12. The view of the DEVS models

Execution of the DEVS models is done by an input message injector GUI with which a user enter the values of variables. The number of rows is determined by extracting the information of the input data type from the schema in the WSDL. After setting the values in the VALUE column, the inject value button makes the XML message in accordance with the data type described in the schema. The injecting message goes through the input port of the DEVS coupled model. The output message is displayed in the *show the result* table.

6. CONCLUSION

DEVSI4WS system is introduced in this paper to consider integration of business processes as DEVS modeling and simulation through mapping web services to DEVS atomic models. When business processes are converted to DEVS models, the models can be operable to any domain DEVS models which can be used as observer model called an experimental frame [14]. Experimental frame can provide dynamic testing environment to the business processes.

To connect a web service to a web service, DEVSI4WS applies message inclusion scheme based on WSDL. The scheme compares input port message type to output port message type and couples two web services when the input port message includes the output port message.

DEVS atomic models generated by mapping mechanism encapsulate a dynamic invocation method of a web service. The method requires XML message input called a SOAP body and produces a SOAP message from the web service. The SOAP body is created using conversion function.

The integration of DEVSI4WS is written in a XML document which is used as input during execution of the integration of DEVSI4WS. The XML document provides distributed testing of business processes over DEVS/SOA environment which is able to simulate DEVS models over service oriented architecture environment [20-23].

As future works, implementation of work flow in the DEVSI4WS is required to cover the whole capability of BPEL functions such as activities and structuring activities. Some work flow components is easily implemented in the DEVS modeling, but some is needed to employ new concept to implement. The message inclusion should be upgraded to cover more complex messages from web services and element messages from work flow component models. To create distributed test environment for business processes with experimental frame concept, we need to revise DEVS/SOA environment to add interpreter of XML document for DEVSI4WS and generator of DEVS models from the XML document. Also, to integrate DEVS models for business processes and experimental frame models, we need to modify the XML document to accommodate the experimental frame models.

References

[1] SOA http://www.sun.com/products/soa/index.jsp
[2] Web Service Architecture http://www.w3.org/TR/ws-

arch/
[3] WSDL2.0 http://www.w3.org/TR/wsdl20-primer/
[4] SOAP1.2 http://www.w3.org/TR/soap12-part0/
[5] Martin Owen and Jog Raj, “BPMN and Business Process Management Introduction to the New Business Process Modeling Standard”, Popkin Software, 2003.
[6] OMG, “Business Process Modeling Notation Specification”, 2006
[7] Stephen A. White, “Using BPMN to Model a BPEL Process”, IBM, February 2005.
[8] ActiveBPEL : http://www.activebpel.org/
[9] ParasoftBPEL : www.parasoft.com/
[10] Oracle : www.oracle.com/technology/bpel
[11] IBM : www.ibm.com/developerworks/library/ws-bpel/
[12] WSUnit : https://wsunit.dev.java.net/
[13] S. Mittal, “DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures”, PhD Dissertation, University of Arizona, 2007S. Mittal, “DEVS Unified Process for Integrated
[14] Zeigler, B.P., Kim, T.G., and Praehofer, H., Theory of Modeling and Simulation, 2nd ed., Academic Press, New York, 2000.
[15] XML http://www.w3.org/XML/
[16] Michael Champion, Chris Ferris, Eric Newcomer, and David Orchard, “Web Services Architecture”, W3C, 2002.
[17] JAXB, http://java.sun.com/developer/technicalArticles/WebServices/jaxb/
[18] Apache AXIS2 : http://ws.apache.org/axis2/
[19] DEVSJAVA : http://www.acims.arizona.edu/
[20] Mittal, S., Risco-Martín, J.L., Zeigler, B.P.,"Implementation of Formal Standard for Interoperability in M&S/Systems of Systems Integration with DEVS/SOA", submitted to C2 Journal
[21] Saurabh Mittal, Bernard P. Zeigler, Jose L. Risco Martin, Jesús M. de la Cruz,"WSDL-Based DEVS Agent for Net-Centric Systems Engineering", presented at the MAS 2008, Italy
[22] Saurabh Mittal, Bernard P. Zeigler, DEVS Unified Process for Integrated Development and Testing of System of Systems, Critical Issues in C4I, AFCEA-George Mason University Symposium, May
[23] Mittal, S., Risco-Martin, J.L., Zeigler, B.P., "DEVS-Based Simulation Web Services for Net-centric T&E", Summer Computer Simulation Conference SCSC'07, July

Biography

Chungman Seo is a Ph.D. student in Electrical & Computer Engineering (ECE) at the University of Arizona and a member of ACIMS. His research interests include DEVS based web service integration; DEVS/SOA based distribution DEVS simulation, and DEVS simulator interoperability.

Bernard P. Zeigler, Ph.D. is Professor of Electrical and Computer Engineering at the University of Arizona, Tucson and Director of ACIMS. He is internationally known for his 1976 foundational text *Theory of Modeling and Simulation*, revised for a second edition (Academic Press, 2000), He has published numerous books and research publications on the Discrete Event System Specification (DEVS) formalism. In 1995, he was named Fellow of the IEEE in recognition of his contributions to the theory of discrete event simulation.