Interoperability between DEVS Simulators using Service Oriented Architecture and DEVS Namespace

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, Service Oriented Architecture, SOAP, WSDL, Interoperability of DEVS simulators, DEVS namespace

Abstract

DEVS Modeling and Simulation (M&S) has various implementations with various computer languages such as JAVA, C++, and C#. To enhance model reusability with different implementation, we need interoperable systems such as CORBA, HLA, and SOA, and an interoperable mechanism for simulation of heterogeneous DEVS models. As an infrastructure of an interoperable system, SOA is applicable because it provides platform and language independence. In this paper, we apply DEVS/SOA system to embody interoperability between heterogeneous DEVS models. Also, we define common DEVS simulator interface required to simulate DEVS models. To expose simulators' ports information described in XML schema type, the interface has additional description operations. The XML schema type is registered into DEVS namespace which is common place to look up the schema type for ports when heterogeneous DEVS models are integrated through DEVS simulation service integration/execution which we propose. We will illustrate the example of DEVS simulator interoperability with DEVSJAVA and ADEVS models which is implemented with JAVA and C++, respectively

1. INTRODUCTION

Interoperability between heterogeneous software systems is an important issue to increase software reusability in the software industry. Many methods are proposed to implement interoperable systems using distributed infrastructures such as CORBA, HLA and SOA [1-3]. Those infrastructures can communicate between software systems with different languages. SOA provides more flexible approach to interoperability than others because it provides platform independence and employs XML message called Simple Object Access Protocol (SOAP) to communicate between a server and a client [4-7].

DEVS research community tries to develop the interoperable framework to simulate DEVS models generated in the different languages and system platforms. Some researches on interoperability on DEVS have been studied along with CORBA and SOA. These researches apply simulator level interoperability that uses common simulator interface to simulate DEVS models. The simulator interface describes a minimum agreement being able to implement a simulator instance using different languages such as JAVA, C++, and C#. This approach strengthens model reusability because DEVS Modeling and Simulation separates models and simulator.

In this paper, we propose interoperability system using DEVS/SOA and DEVS namespace. Web services represent DEVS simulators embedding specific DEVS models. They have minimum agreement for simulator and information of in/out ports which have specific data types described in DEVS namespace. Proposed system helps increase DEVS model composability because of easy integration of DEVS models if information of input port is same as that of output port when DEVS models are coupled.

To implement the interoperability system, we define Web Service Description Language (WSDL) for the web service and create DEVS namespace holding type definitions. Before web services created by common simulator WSDL are published, the data types of ports of models are registered in the DEVS namespace.

In the rest of the paper, the background of DEVS and SOA is discussed in the section 2. The section 3 addresses an overall system of interoperability of DEVS simulators, structure of DEVS simulation service, and DEVS simulation service integration and execution. The section 4 explains implementation of interoperability of DEVS simulators. In the section 4, we demonstrate two web services using JAVA and VC++ with DEVSJAVA and ADEVS, respectively. The example of integration of DEVS web services is presented in the section 4. The paper's summary and future works are in the section 5.

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. 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 [8]. 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 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) [7]. 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 [5] 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) [6] 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.

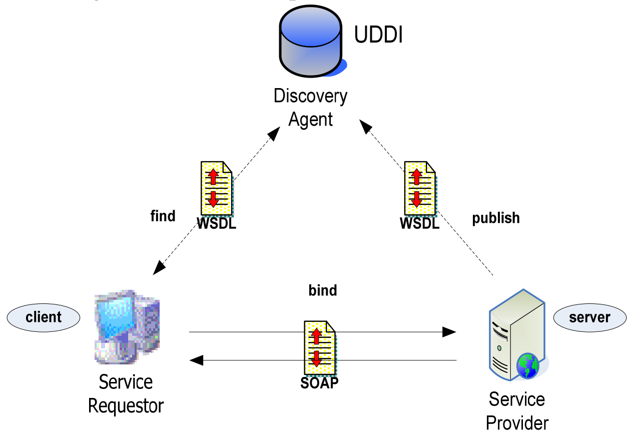


Figure 1 Web Services Architecture

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) [4], 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 SYSTEM OF INTEROPERABILITY OF DEVS SIMULATORS

The interoperability system of DEVS simulators consists of three parts which are a DEVS namespace, DEVS simulation services, and DEVS simulation service integration and execution (DSSIE). The DEVS namespace is a schema that contains message type definitions. It is used to recognize message types between distributed or different systems when they need to cooperate in system of systems. Each service registers its message types in the DEVS namespace before it publishes in the UDDI.

The DEVS simulation service has a common interface to provide interoperability between different platforms or different languages. The common interface is called WSDL defining operations, message types, and the location of the service. The DEVS simulation service can be implemented on various operating systems and computer languages. However, SOAP messages as requests and responses of operations provide loosely coupled distributed computing and neutral message passing. In the figure 2, the two DEVS simulation service is connected to common interface on the different platforms.

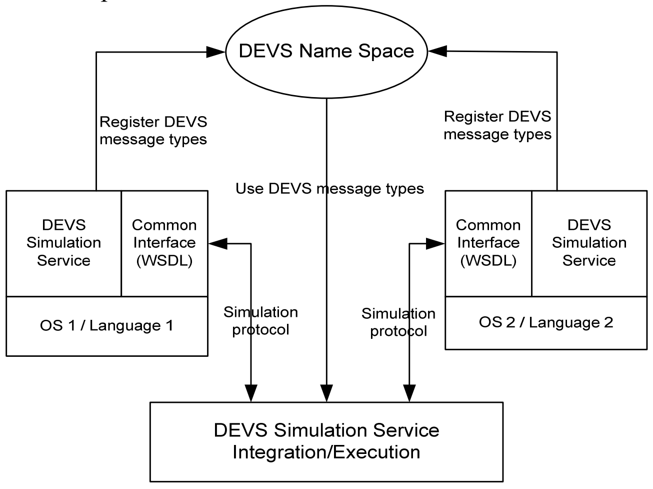


Figure 2 Overall system of DEVS simulator interoperability

DSSIE has two functions which are the integration of the DEVS simulation services based on message types and the execution of the integrated system. The integration of the DEVS simulation services is performed by a GUI called a DEVS simulation service integrator which uses the DEVS namespace to verify if couplings between two services are possible or not. The data on the integrator are written to a XML document sent to the executor which simulates DEVS simulation services. The executor adopts Java Architecture for XML Binding (JAXB) API to makes handling the XML easy. In the figure 2, the DSSIE obtains DEVS message types of DEVS simulation services from the DEVS namespace to integrate services and simulate the DEVS services with simulation protocols.

3.1. The Structure of DEVS Simulation Service

To implement SOA for DEVS simulation, a middleware for helping create the web service is needed such as Apache eXtensible Interaction System (AXIS2) [9] or .Net framework [12]. The middleware provide API to make building a web service easy, hiding complicating network programming. AXIS2 is adopted for web service middleware which is embodied by Java program while .Net can be used for C++ or C# user.

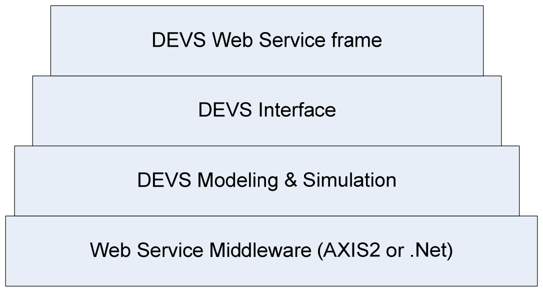

DEVS Web Service frame
DEVS Interface
DEVS Modeling & Simulation
Web Service Middleware (AXIS2 or .Net)

Figure 3. The structure of DEVS web service implementation

The figure 3 represents that DEVS web service is supported by several APIs to implement a DEVS simulator interoperability system. The bottom layer is web service middleware which manipulates network connection and SOAP messages between a web service and a client program. DEVS modeling and simulation API enable DEVS modeling and simulation to be used in the web service. The role of DEVS interface is connection between web service frame which is described by WSDL, and DEVS modeling and simulation.

In the DEVS simulator interoperability environment, DEVS web service frame has a WSDL whose name is *Simulation* providing three interfaces which are the interface of DEVS protocols, the interface of the reporting function, and the interface of information of message location and types.

The interface of DEVS protocols in 1 of the figure 4 is utilized when DEVS simulation starts. There are seven operations, that is, *initialize, getTN, lambda, getOutput, receiveInput, deltfcn, and exit*. The *initialize* and *exit* operations are called when the simulation starts and ends. The *getTN* operation returns next internal event time (TN) to a coordinator which is in the DEVS simulation service integration and execution on figure 2. The *lambda* operation generates output messages if its own model has an internal event. The *getOutput* operation returns output messages which are consist of XML document to the coordinator which looks up the coupling table and request the invocation of the *receiveInput* operation to the corresponding DEVS simulation service. Thereafter, the *deltfcn* operation changing the state of the model and scheduling TN is called to all DEVS simulation services. This is one cycle of DEVS simulation protocol. The simulation protocol is repeated until meeting the certain condition to stop the simulation such as infinity of TN of all simulation services, and the number of simulation protocol cycles.

The interface of the reporting function in 2 of the figure 4 has two operations, that is, *getConsole* and *getResult*. The getConsole operation returns the output produced by the simulation service during simulation protocol cycles. With the output, a user can validate if the model in the simulation service is appropriately working. The *getResult* operation returns the result of the simulation if the simulation service generates data written in the result document located in the specific place

```
public interface Simulation{
    public void initialize(double t);
    public double getTN();
    public void lambda(double t);
    public String getOutput();
    public void receiveInput(String               1
      portFrom, String msg, String
      portTo);
    public void deltfcn(double t);
    public void exit();
    public String getConsole(String
      clientIp);                                  2
    public String getResult();
    public String getSchemaInfo();
    public String getType(String
      port);
    public void getSimulator(boolean
      isRT);
    public String[] getInports();                 3
    public String[] getOutports();
    public void addCoupling(String
      portFrom, String portTo, String
      ipServiceTo);
  }
```

Figure 4. The operations of Simulation service.

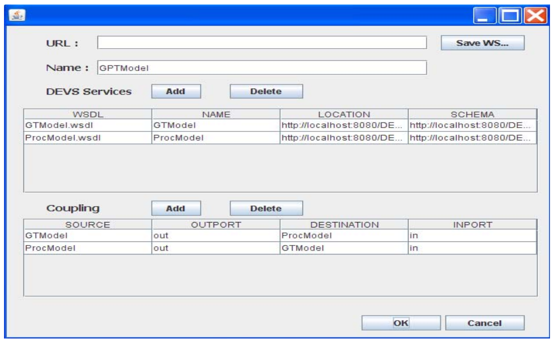
The interface of the information of message location and types in 3 of the figure 4 has five operations which are *getSchemaInfo*, *getType*, *getInports*, *getOutports*, and *getSimulator*. Each simulation service registers its message types in the schema repository called DEVS namespace and exposes the location of the schema, the name of input ports and output ports, and message types through the four operations except the *getSimulator* operation which is used when a simulator is selected if there are several simulators such as centralized or parallel mode in the simulation service. For example, if we want to simulate DEVS models with real time, the argument of the *getSimulator* is set to true. The real time simulator is used with the DEVS model in the web service.

3.2. DEVS Simulation Service Integration and Execution

The GUI of DSSIE consists of four elements which are WSDL handler, name of integration, DEVS service handler, and coupling handler as shown in the figure 5. The WSDL handler saves on a specific place a selected WSDL which is used in the integration. After the WSDL name of DEVS service is written in the textfield by the URL label, pushing the save WSDL button writes the WSDL from the URL address. The name of integration provides a title of a XML document.

The DEVS service handler begins with clicking add button by a DEVS services label. The information of service GUI shows up immediately. The GUI displays a repository where WSDLs are saved. The selected WSDL from the repository provides a WSDL file name, a model name, WSDL location, and schema location obtained by invocation of an operation called *getSchemaInfo* in the figure 4. That information is displayed on the table below the DEVS services label.

After selecting some DEVS simulation services in the DEVS service handler, the coupling handler is carried out. Pushing the add button by the coupling label shows a GUI for helping make coupling between DEVS simulation services. The GUI displays a source, an output message, a destination, and an input message if the output message is equal to the input message. When displaying an output message and an input message, the invocations of three operations are performed to get the output ports and input ports, and the message type of each port. The operations are *getInports*, *getOutports*, and *getType* in the figure 4. The coupling information is shown in the table below coupling label.



URL :

Name : GPTModel

DEVS Services

WSDL	NAME	LOCATION	SCHEMA
GTModel.wsdl	GTModel	http://localhost:8080/DE...	http://localhost:8080/DE...
ProcModel.wsdl	ProcModel	http://localhost:8080/DE...	http://localhost:8080/DE...

Coupling

SOURCE	OUTPORT	DESTINATION	INPORT
GTModel	out	ProcModel	in
ProcModel	out	GTModel	in

Figure 5. The DEVS simulation service integrator GUI

After finishing the integration of DEVS simulation services, clicking the ok button creates a XML document structured to contain the information from the integrator. The schema for the XML document is defined to validate an instance of the schema. The XML document begins with a *devswsintegrator* tag, and has five tags, that is, *title, services,* *couplinginfo*, *inports*, *and outports*. The *services* tag can have many *model* tags which have *wsdl*, *name*, *location*, and *schema* tags. Similarly, *coupinginfo* tag can have many *coupling* tags which have *source*, *outport*, *destination*, and *inport* tags.

A coordinator prepares the simulation of the integration using the XML document, and executes the simulation. The preparation of the simulation includes making instances of client proxies for DEVS simulation services and structuring coupling information. The execution of the simulation adopts a centralized method which controls simulation protocols in the coordinator. Figure 6 represents the centralized simulation protocol that displays calling operations in the coordinator to the DEVS simulation services. The coordinator has the instances of client proxies for the simulation services. This simulation begins with calling an *initialize* operation to all simulation services. After that, the coordinator requests a *getTN* operation to get next time for an internal event of a DEVS model. Calculating a minimum time of next times, a *lambda* operation is called with an argument which is the minimum time. After the lambda operation, the simulation service with minimum time produces an output message. The response of the *getOutput* is output messages of the simulation services. The coordinator looks up the coupling information which displays the flow of messages to inject the output message to a corresponding simulation service. Through a *receiveInput* operation, the output message is sent to the simulation service which has the corresponding DEVS model. After routing the output message, the coordinator requests a *deltFunc* operation to all simulation services to execute an internal or external event function in the simulation service. The coordinator repeats this procedure except the initialization of the simulation service until meeting a certain condition to terminate the simulation.

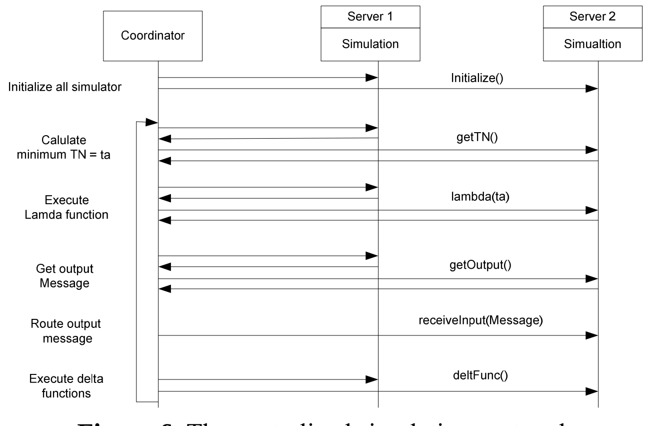


Figure 6. The centralized simulation protocol

4. IMPLEMENTATION OF INTEROPERABILITY OF DEVS SIMULATORS

DEVS modeling and simulation is implemented with Java, C++, or C# languages according to intention of the designers. To demonstrate the concept of interoperability using DEVS Simulators, two DEVS M&S instances implemented with different computer languages and system environments are used. One is DEVSJAVA [10] developed with Java language by ACIMS lab, and the other is ADEVS [11] embodied with C++.

4.1. Web Services encapsulating DEVSJAVA

DEVSJAVA consists of three libraries, that is, DEVS M&S supporting data structures, modeling, and simulation. All libraries follow an object oriented design concept which presents inheritance, polymorphism, and information hiding. The modeling library is used to create two kinds of DEVS models which are atomic and coupled models. The simulation library helps execute DEVS models. There are two main classes called a coordinator and a simulator. The simulator controls an atomic model and the coordinator manages the simulators through the message passing such as simulation time and DEVS messages. The data structures support modeling and simulation with holding necessary objects such as models, simulators, and messages.

DEVS coupled model has a hierarchical structure and each model is encapsulated by a coordinator or a simulator. There is a top level coordinator which decides minimum time advance from all coordinators and simulators which send time advance, and controls simulation protocols. Figure 7 represents assignment of a simulator or a coordinator to each model in a top level coupled model [13]. The simulators, coordinators and DEVS models are in the same machine. DEVS message is passed with a putMessage function among the same level components. There are two functions to send messages to a lower level or an upper level, that is, sendDownMessage and putMyMessage.

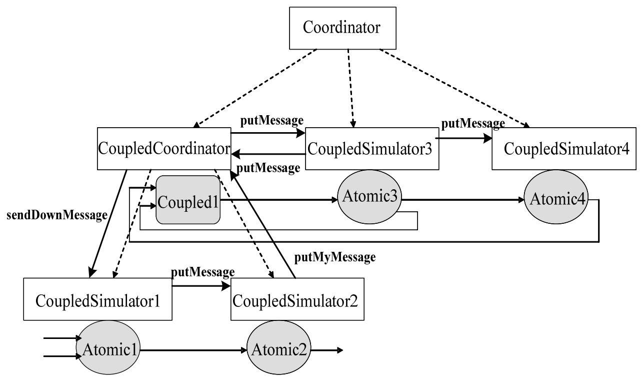


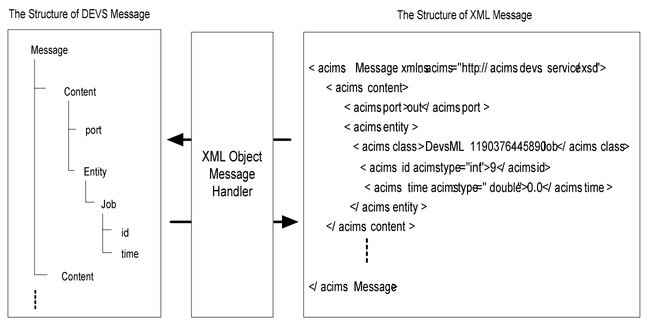
Figure 7. The view of relationship between a model and a simulator or a coordinator

DEVS message can be any object inherited by an entity class. A container for DEVS message is a content class which has two variables representing a port of a model and a DEVS message. To cover multiple content classes, there is a container for contents which is called a message class.

To produce web services for DEVSJAVA, apache AXIS2 which is a framework for developing XML based web service with JAVA or C++ is used. AXIS2 provides several tools such as java2WSDL, service archiver and code generator to help easy create web services using JAVA. The java2WSDL generates a WSDL from a java code, and the code generator creates JAVA codes for server side codes or client side codes which include a mechanism of a conversion of a JAVA object to a SOAP message and vice versa. The service archiver turns classes of JAVA codes to a web service archive file which runs in the apache tomcat server.

The operations of the web service of DEVSJAVA are similar to the functions of the simulator in the DEVSJAVA. It is true if a DEVS model is an atomic but not a coupled model. For a coupled model to pretend to have the functions of an atomic model, a Digraph2Atomic class which includes minimum functions for simulation is used. Therefore, an atomic or coupled model can be a web service with a DEVS interface library.

It is important to have neutral message for interoperability between heterogeneous models. But DEVS message can be created with any object. In JAVA language, there is object serialization to send a JAVA object to other place connected to the network. JAVA serialization is not applicable to interoperable systems. So, DEVS message need to be converted to XML based message being able to solve this problem.


The Structure of DEVS Message
Message
Content
port
Entity
Job
id
time
Content
XML Object Message Handler
The Structure of XML Message

```
< acims  Message xmlns:acims="http:// acims devs  service/xsd">
   < acims  content>
      < acims port >out</ acims port >
      < acims entity >
        < acims class>DevsML  1190376445890Job</ acims class>
          < acims  id acimstype="int">9</ acims id>
           < acims  time acimstype=" double">0.0</ acims time >
        </ acims entity >
   </ acims content >

</ acims  Message>
```

Figure 8. Example of XML Object Message Handler

XML Object message handler is employed to transform an object DEVS message to a XML-style message. As seen in the figure 8, the structure of the message container consists of more than one content classes containing port and entity object. Entity object can be any type of objects inheriting an entity class in the DEVS library. XML-style message consists of tags representing the message container. Tag names are “Message”, “content”, “port”, and “entity”. And entity tag has “class” and the names of variables of DEVS message. The XML-style message is passed to other web service and XML Object message handler converts XML-style message to JAVA Object.

4.2. Web Service encapsulating ADEVS

ADEVS is implementation of DEVS based on parallel and dynamic DEVS formalism using C++ [11]. It consists of modeling, simulation, and container libraries. The modeling library can create atomic and coupled models like DEVSJAVA. The simulation library has one simulator class which has a scheduler and controls simulation protocol. To get the minimum time advance (TA), the simulator uses the scheduler with a bag container storing atomic models. That is, ADEVS does not use hierarchy structure of model when calculating TA. When passing DEVS messages from a source model to a destination model, coupling information in the coupled model is needed. The container library is a bag class which holds any type of object.

The simulation of ADEVS model is executed by three functions in the simulator class. The functions are the following.

- nextEventTime() calls a scheduler’s minPriority() calculating next event time and classifying imminant models which have next event time and a token for internal transition.
- computeNextOutput() executes a lamda function if any model has next event time and route the output message to destination models which have a token for external transition.
- computeNextState() executes a delta function according to the token which the models have and initialize the tokens and input/output message containers of models..

To make web service for ADEVS, visual studio VC++ is used in the .Net environment. The web server is provided by windows OS such as windows XP or windows Vista. The visual studio has a template to create web services [12]. With the template, the operations of the interoperable simulator service are declared like the figure 4. The operations should be connected to the ADEVS simulator to pass simulation protocol values and DEVS messages to the client. However, the ADEVS simulator does not cover interoperable simulator operations. To meet this end, the modified ADEVS simulator is introduced.

The modified simulator adds getTN(), lamda(), getOut(), putMessage(), and deltfcn() through separating three functions. The getTN() calculates next event time from the models, lamda() generates output messages if any model has next event time, getOut() sends output message to outside of the simulator, putMessage() injects input messages into the simulator, and deltfcn() executes internal and external transitions if applicable to models and initializes input/output message containers and the token for transitions.

The ADEVS message can be any object, and PortValue class has port and ADEVS message. Message container is a bag class containing PortValue instances. As the DEVSJAVA, the message container is converted to XML-style message to provide an interoperability environment. The getOut operation has XML conversion mechanism to create XML-style message. The receiveInput operation has c++ object conversion mechanism to make an input container.

4.3. DEVS Simulator Web Services and Integration

To demonstrate DEVS interoperability system, an example DEVS model called GPT is used as seen in figure 9. The GPT model consists of a coupled model called an Experiment Frame (EF) model and an atomic model called a *Processer* model. The EF model has two atomic models called *Generator* and *Transducer*. The GPT model uses Job type message which has two variables, that is, id and time. The Generator creates new Job type message repeatedly according to internal time of the model. The Processer processes the Job coming from in port. If the Job is finished, it is sent to the Transducer which collects information of generated or processed Jobs and takes the statistics during a certain time. If the certain time is passed, the Transducer sends the message to the Generator to stop generating a Job message.

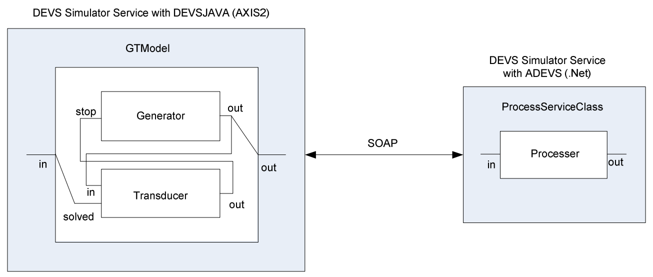
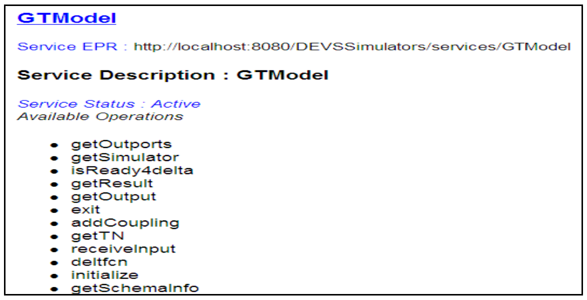


Figure 9. The view of the GPT model consisting of two web services

Two web services are generated to simulate the GPT model using the interoperability system. One is created with a JAVA based system and DEVSJAVA. The web service name is GTModel containing the EF model as seen in figure 10. The JAVA based system is Apache AXIS2 operated on Apache tomcat server.


GTModel

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

Service Description : GTModel

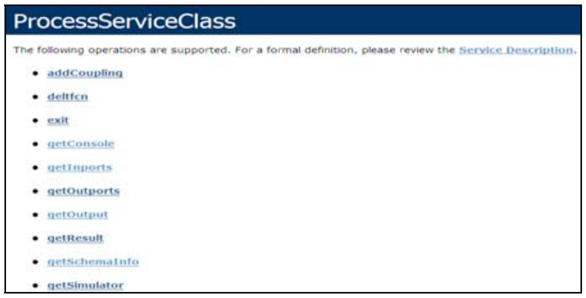
Service Status : Active
Available Operations

- getOutports
- getSimulator
- isReady4delta
- getResult
- getOutput
- exit
- addCoupling
- getTN
- receiveInput
- deltfcn
- initialize
- getSchemaInfo

Figure 10. GT simulation web service using AXIS2 and DEVSJAVA

The other web service is generated with Visual C++ based system and ADEVS. ProcessServiceClass is the name of the web service embedding the Processor model as seen in figure 11.

To integrate two different web services, DEVS web service integrator is used as shown in figure 5. It creates a XML document which contains information of the location of the web service and coupling of selected web services. Figure 12 shows the XML document example of integration of EF and Processer web services.


ProcessServiceClass

The following operations are supported. For a formal definition, please review the Service Description.

- addCoupling
- deltfcn
- exit
- getConsole
- getInports
- getOutports
- getOutput
- getResult
- getSchemaInfo
- getSimulator

Figure 11. Process Simulator web service using .Net and ADEVS

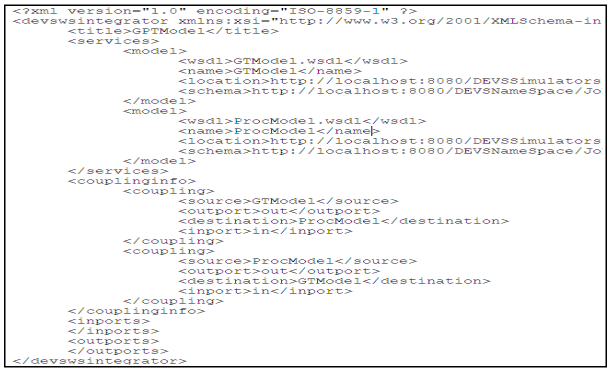
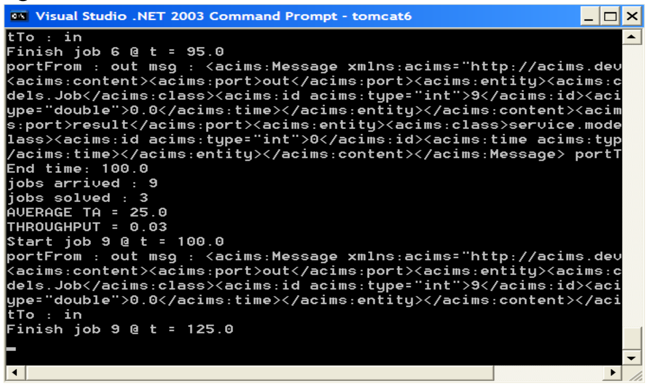

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<devswsintegrator xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
        <title>GPTModel</title>
        <services>
                <model>
                        <wsdl>GTModel.wsdl</wsdl>
                        <name>GTModel</name>
                        <location>http://localhost:8080/DEVSSimulators
                        <schema>http://localhost:8080/DEVSNameSpace/Jo
                </model>
                <model>
                        <wsdl>ProcModel.wsdl</wsdl>
                        <name>ProcModel</name>
                        <location>http://localhost:8080/DEVSSimulators
                        <schema>http://localhost:8080/DEVSNameSpace/Jo
                </model>
        </services>
        <couplinginfo>
                <coupling>
                        <source>GTModel</source>
                        <outport>out</outport>
                        <destination>ProcModel</destination>
                        <inport>in</inport>
                </coupling>
                <coupling>
                        <source>ProcModel</source>
                        <outport>out</outport>
                        <destination>GTModel</destination>
                        <inport>in</inport>
                </coupling>
        </couplinginfo>
        <inports>
        </inports>
        <outports>
        </outports>
</devswsintegrator>
```

Figure 12. The XML document for DEVS Simulator WS Integration

To execute the XML document, an execution program is called a coordinator which prepares simulation and runs the simulation. The procedure of the simulation follows figure 6.



```
tTo : in
Finish job 6 @ t = 95.0
portFrom : out msg : <acims:Message xmlns:acims="http://acims.dev
<acims:content><acims:port>out</acims:port><acims:entity><acims:c
dels.Job</acims:class><acims:id acims:type="int">9</acims:id><aci
ype="double">0.0</acims:time></acims:entity></acims:content><acim
s:port>result</acims:port><acims:entity><acims:class>service.mode
lass><acims:id acims:type="int">0</acims:id><acims:time acims:typ
/acims:time></acims:entity></acims:content></acims:Message> portT
End time: 100.0
jobs arrived : 9
jobs solved : 3
AVERAGE TA = 25.0
THROUGHPUT = 0.03
Start job 9 @ t = 100.0
portFrom : out msg : <acims:Message xmlns:acims="http://acims.dev
<acims:content><acims:port>out</acims:port><acims:entity><acims:c
dels.Job</acims:class><acims:id acims:type="int">9</acims:id><aci
ype="double">0.0</acims:time></acims:entity></acims:content></aci
tTo : in
Finish job 9 @ t = 125.0
_
```

Figure 13. The result of simulation

The simulation result is shown in the figure 13. It is the same as that of the GPT model simulated in the DEVSJAVA.

5. CONCLUSION

In this paper, we designed and implemented DEVS simulator interoperability system with service oriented architecture and DEVS namespace. To integrate heterogeneous DEVS simulator web services, we developed a DEVS simulator web service integrator which extracts information from the simulator web services to verify if two simulator web services can be coupled. A XML document is used to execute integrated web services.

In this system, DEVS messages are converted to XML-style messages to be interoperable in the different language and platform. Dynamic converter of JAVA object to XML message is implemented in the paper. But the converter does not cover all possible JAVA objects. Neither does the converter in C++.

ADEVS library does not cover some simulator web service operations because it does not provide those functions. We make an interface simulator to connect the ADEVS simulator and simulator web service operations. Through the interface simulator, ADEVS model can be simulated with a simulation protocol in figure 6.

As future works, dynamic message conversion mechanism should be developed with generic approaches. Other language version of DEVS needs to be developed to cover all language implementation of DEVS. This system need to test a real system requiring interoperability between legacy and new system, and evaluate the performance of DEVS simulator web service system [14].

References

[1] Wutzler, T. H.S. Sarjoughian (2007), “Interoperability among Parallel DEVS Simulators and Models Implemented in Multiple Programming Languages”, SIMULATION: Transactions of The Society for Modeling and Simulation International, Accepted.

[2] Sarjoughian, H. S., and B. P. Zeigler. "DEVS and HLA: Complementary Paradigms for Modeling and Simulation?" *Simulation: Transactions of the Society for Modeling and Simulation International* 17, no. 4 (2000): 187-97.

[3] Mittal, S., and J. L. R. Martín. "DEVSML: Automating DEVS Execution over SOA Towards Transparent Simulators Special Session on DEVS Collaborative Execution and Systems Modeling over SOA." Paper presented at the DEVS Integrative M&S Symposium DEVS' 07 2007.

[4] SOA http://www.sun.com/products/soa/index.jsp

[5] Web Service Architecture http://www.w3.org/TR/ws-arch/

[6] WSDL2.0 http://www.w3.org/TR/wsdl20-primer/

[7] SOAP1.2 http://www.w3.org/TR/soap12-part0/

[8] Zeigler, B.P., Kim, T.G., and Praehofer, H., Theory of Modeling and Simulation, 2nd ed., Academic Press, New York, 2000.

[9] Apache AXIS2 : http://ws.apache.org/axis2/

[10] DEVSJAVA : http://www.acims.arizona.edu/

[11] ADEVS: an open source C++ DEVS Simulation engine. Available at: http://www.ornl.gov/~1qn/adevs/index.html

[12] Microsoft Corporation. XML and .NET White Papers. http://www.microsoft.com/serviceproviders/whitepapers/xml.asp

[13] Xiaolin Hu, Bernard Zeigler, " A Proposed DEVS Standard: Model and Simulator Interfaces, Simulator Protocol"

[14] 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

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.