WSDL-BASED DEVS AGENT FOR NET-CENTRIC SYSTEMS ENGINEERING

Saurabh Mittal[(a)], Bernard P. Zeigler[(b)], Jose L. Risco Martin[(c)], Jesús M. de la Cruz[(c)]

[(a)]DUNIP Technologies, Inc. New Delhi, India
[(b)]Arizona Center for Integrative Modeling and Simulation
ECE Department, University of Arizona, Tucson, AZ USA
[(c)] Departamento de Arquitectura de Computadores y Automática
Facultad de Informática, Universidad Complutense de Madrid, Madrid, Spain

[(a)]saurabh.mittal@duniptechnologies.com, [(b)]zeigler@ece.arizona.edu, [(c)] jlrisco@dacya.ucm.es, jmcruz@fis.ucm.es

ABSTRACT
This research work provides a methodology to use Discrete Event Systems Specification (DEVS) to design and evaluate the performance of web services within a Service Oriented Architecture (SOA). We will show how a Web Service Description Language (WSDL) document can be mapped to a DEVS model in an automated manner through a DEVS abstract service wrapper. This work will describe the underlying architecture of abstract DEVS service wrapper and a workflow example made executable using the DEVS/SOA framework. This work will establish DEVS as a production environment for net-centric systems as well as a solid M&S analysis tool for SOA design.

Keywords: SOA, WSDL, DEVS/SOA, DEVSML

1. INTRODUCTION

Industry and government are spending extensively to transition their business processes and governance to Service Oriented Architecture implementations for efficient information reuse, integration, collaboration and cost-sharing. Service Oriented Architecture (SOA) enables orchestrating web services to execute such processes using Business Process Execution Language (BPEL). Business Process Modeling Notation (BPMN) is another method that outputs BPEL for deployment. As an example, the Department of Defense's (DoD grand vision is the Global Information Grid that is founded on SOA infrastructure. As illustrated in Figure 1, the SOA infrastructure is to be based on a small set of capabilities known as Core Enterprise Services (CES) whose use is mandated to enable interoperability and increased information sharing within and across Mission Areas, such as the Warfighter domain, Business processes, Defense Intelligence, and so on) (DoD GIG Architecture, 2007). Net-Centric Enterprise Services (NCES) is DoD's implementation of its Data Strategy over the GIG. NCES provide SOA infrastructure capabilities such as service and metadata registries, service discovery, user authentication, machine-to-machine messaging, service management, orchestration, and service governance.

However, composing/orchestrating web services in a process workflow (a.k.a Mission thread in the DoD domain) is currently bounded by the BPMN/BPEL technologies. Moreover, there are few methodologies to support such composition/orchestration. Further, BPMN and BPEL are not integrated in a robust manner and different proprietary BPMN diagrams from commercial tools fail to deliver the same BPEL translations. Today, these two technologies is by far the only viable means whereby executives and managers can devise process flows without touching the technological aspects. With so much resting on SOA, their reliability and analysis must be rigorously considered. The BPMN/BPEL combination neither has any grounding in system theoretical principles nor can it be used in designing net-centric systems based on SOA in its current state.

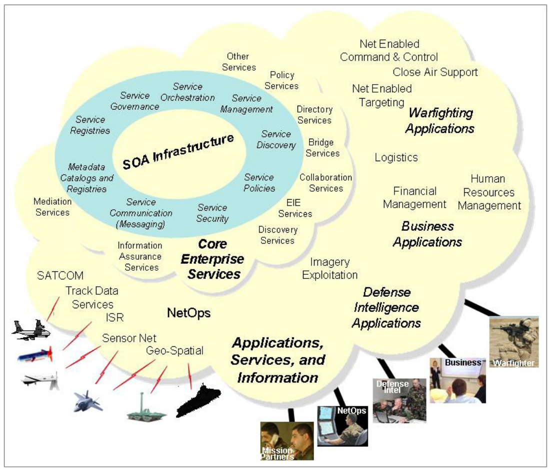


Figure 1: Core Enterprise Services in GIG

In this research work we provide a proof of concept of how Discrete Event System Specification (DEVS) Formalism can deliver another process work flow mechanism to compose web services in a SOA. We will show how the resulting agent based DEVS system can be executed on the recently developed DEVS/SOA (DUNIP, 2007; Mittal, Martin and Zeigler, 2008) distributed modeling and simulation framework. In addition to supporting SOA application development the framework enables verification and validation testing of application. The developed DEVS models from WSDL lie in the subset of DEVS specifications known as Finite Deterministic DEVS or FDDEVS (Hwang and Zeigler, 2006; Mittal, Zeigler and Hwang, 2007) that can be used for verification. However, V&V is not the focus of this paper. We will demonstrate the execution of the DEVS agent in a complete case-study in which a workflow is composed and executed using

DEVS/SOA framework. The paper is organized as follows. Section 2 presents the related work in the area of BPEL, BPMN and Agent based studies focused towards SOA. Section 3 describes the underlying technologies that include DEVS, Web Services framework. Section 4 deals with Abstract DEVS Service wrapper in detail and also discusses how statistics gathering is integrated with the wrapper design. Section 5 presents the implementation of DEVS WSDL agent and how it can be used in a process workflow using the proposed Web Services Workflow Formalism, WSWF. Section 6 presents layered architecture of Agent-based Test Instrumentation System on/using Global Information Grid using SOA (GIG/SOA) that provides a larger perspective on the application of DEVS-WSDL agent architecture. Finally, Section 7 presents conclusions and future work.

2. RELATED WORK

In 2003 there were more than 10 recognized groups defining standards for BPM related activities, 7 of these bodies were working on modeling definitions so it's no wonder that the whole picture got very confused (Pyke, 2007) Fortunately there has been a lot of consolidation, and currently only 3 key standards to really take notice:

1. BPMN
2. XPDL
3. BPEL

The Business Process Modeling Notation (BPMN) is a standardized graphical notation for graphically representing business processes workflows. BPMN's primary goal is to provide a standard notation that is readily understandable by all business stakeholders. Stakeholders in this definition include business analysts, technical developers and business managers. BPEL is an "execution language" the goal of which is to enable definition of web service orchestrations. Ultimately, BPEL is all about bits and bytes being moved from place to place and manipulated. XPDL is described not an executable programming language like BPEL, but specifically a process design format that literally represents the "drawing" of the process definition. XPDL is effectively the file format or "serialization" of BPMN. More generally, it can also support any design method or process model that uses the XPDL meta-model. XPDL is a proven format for process design interchange, and it is the most practical standard for establishing a Process Design Ecosystem.
Summarizing, currently there is no popular means other than BPMN/BPEL to design a web service workflow orchestration.

3. UNDERLYING TECHNOLOGIES

This section will give an overview of the technologies used in the development of DEVS Web service M&S framework.

3.1. DEVS

Discrete Event System Specification (DEVS) (Zeigler, Kim and Praehofer, 2000) is a formalism, which provides a means of specifying the components of a system in a discrete event simulation. In DEVS formalism, one must specify *Basic Models* and how these models are connected together. These basic models are called *Atomic Models* (Figure 2a) and larger models which are obtained by connecting these atomic blocks in meaningful fashion are called *Coupled Models* (Figure 2b). Each of these atomic models has *inports* (to receive external events), *outports* (to send events), set of *state variables*, *internal transition*, *external transition*, and *time advance functions*. Mathematically it is represented as 8-tuple system as eq. (1) below:

$$\boldsymbol{M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta\rangle} \quad \textbf{eq. (1)}$$

where,

X is the set of input values
S is the set of state
Y is the set of output values
$\boldsymbol{\delta_{int}}$: S $\rightarrow$ S is the internal transition function
$\boldsymbol{\delta_{ext}}$: Q x X^b $\rightarrow$ S is the external transition function, where
X^b is a set of bags over elements in X,
Q is the total state set.
$\boldsymbol{\delta_{con}}$: S x X^b $\rightarrow$ S is the confluent transition function, subject to $\delta_{con}(s,\Phi) = \delta_{int}(s)$
$\boldsymbol{\lambda}$: S $\rightarrow$ Y^b is the output function
ta: S $\rightarrow$ $R_{0}^{+}{}_{,inf}$ is the time advance function

The model's description (implementation) uses (or discards) the message in the event to do the computation and delivers an output message on the *outport* and makes a state transition.

A DEVS-coupled model designates how atomic models can be coupled together and how they interact with each other to form a complex model. The coupled model can be employed as a component in a larger coupled model and can construct complex models in a hierarchical way. The specification provides component and coupling information. The coupled DEVS model is defined as eq. (2) below:

$$\boldsymbol{M = \langle X, Y, D, \{M_{ij}\},\{I_j\}, \{Z_{ij}\}\rangle} \quad \textbf{eq. (2)}$$

where,

X is a set of inputs
Y is a set of outputs
D is a set of DEVS component names
For each i $\in$ D,
$\mathbf{M_i}$ is a DEVS component model
$\mathbf{I_i}$ is the set of influences for I
For each j $\in$ I_i,

A Java-based implementation of DEVS formalism, DEVSJAVA (Zeigler and Sarjoughian, 2000) can be used to implement these atomic or coupled models.

DEVS formalism consists of models, the simulator and the Experimental Frame as shown in Figure 3. It categorically separates the three of them and they can be perceived of components of a DEVS system architecture. We will focus our attention to the two types of models i.e. atomic and coupled models.

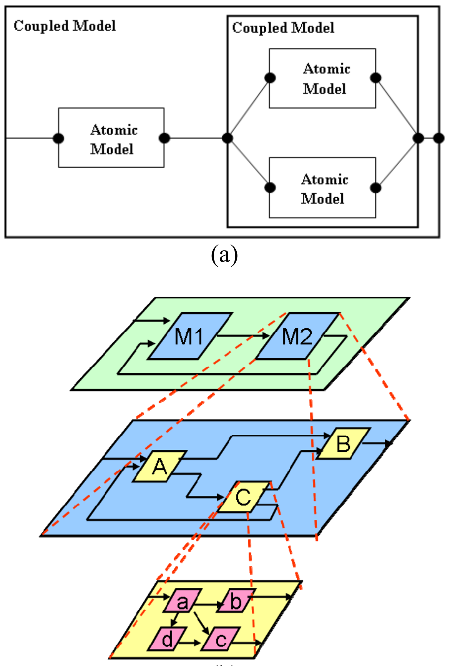


Figure 2: Hierarchical composition of Atomic and Coupled DEVS models.

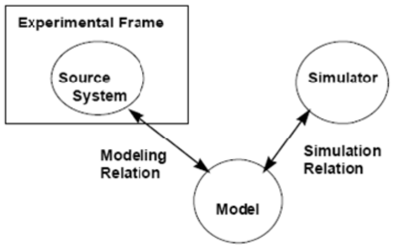


Figure 3: DEVS Separation of Model, Simulator and the Experimental Frame

3.2. Web Services and Interoperability using XML

The Service oriented Architecture (SOA) framework is the orchestration of multiple web services engaged towards a business goal. A Web Service is a component consisting of various W3C standards, in which various computational components are made available as 'services' interacting in an automated manner that achieve machine-to-machine interoperable interaction over the network. The interface is specified using Web Service Description language (WSDL) that contains information about ports, message types, port types, and other relating information for binding two interactions. It is essentially a client server framework, wherein client requests a 'service' using a SOAP message that is transmitted via HTTP in the XML format. A Web service is published by any commercial vendor at a specific URL to be consumed/requested by another commercial application on the Internet. It is designed specifically for machine-to-machine interaction. Both the client and the server encapsulate their messages in SOAP wrappers.

The fundamental concept of web services is to integrate software application as services. Web services allow the applications to communicate with other applications using open standards. To offer DEVS-based simulators as web services, they must have the following standard technologies: communication protocol (Simple Object Access Protocol, SOAP), service description (Web Service Description Language, WSDL), and service discovery (Universal Description Discovery and Integration, UDDI).

4. AN ABSTRACT DEVS SERVICE AGENT

As a crucial part of our workflow, we have designed an Abstract DEVS Service Agent to link DEVS models with Web Services and to generate statistics regarding remote method calls and response times.

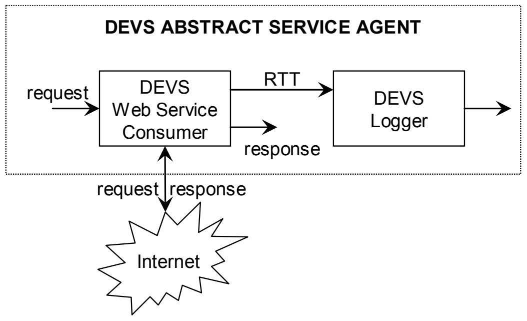


Figure 4: Schematic showing the architecture of our DEVS Agent Service model.

Figure 4 depicts an illustrative example. Our proposed model consists of two DEVS atomic models. The DEVS Web Service Consumer invokes the remote operation provided by means of an external transition. When the operation is processed, this atomic model informs about the round-trip-time (RTT) taken by such operation to the DEVS Logger atomic model as well as the response provided by the Web Service. At the end of the simulation, the DEVS Logger provides statistics such as operations executed successfully, the RTT consumed per operation, etc.

The DEVS Web Service Consumer needs to be configured by means of: (a) the URL of the Web Service, (b) name of the operations offered, and (c) the parameters needed by such operations. This information is specified in the WSDL document.

In order to avoid to the user to extract this information by hand, we have implemented a wrapper which automatically generates the DEVS Web Service Consumer for a Web Service. Thus, given a WSDL address, our framework is able to generate the corresponding DEVS Service Agent.

Web services are utilized using web service clients that are created by various open source and commercially available tools such as Eclipse Web Service Toolkit (WST), Netbeans IDE, Websphere etc.. All of them use the Web Service Description Language (WSDL) as the input to generate the web service client. In our implementation we utilize the Axis2 framework to generate clients. Our choice of Axis2 plugin is driven by the implementation platform of DEVS/framework which is Axis/Java. However, it doesn't matter which method is used to generate the client.

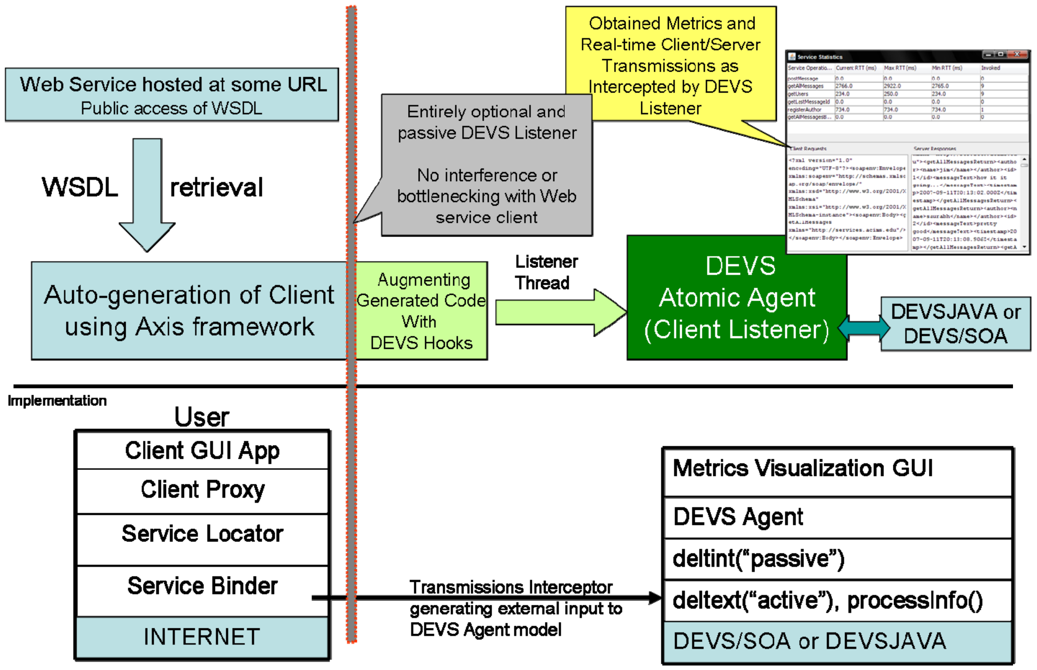

Web Service hosted at some URL
Public access of WSDL
WSDL
retrieval
Auto-generation of Client using Axis framework
Augmenting Generated Code With DEVS Hooks
Entirely optional and passive DEVS Listener
No interference or bottlenecking with Web service client
Obtained Metrics and Real-time Client/Server Transmissions as Intercepted by DEVS Listener
Listener Thread
DEVS Atomic Agent (Client Listener)
DEVSJAVA or DEVS/SOA
Implementation
User
Client GUI App
Client Proxy
Service Locator
Service Binder
INTERNET
Transmissions Interceptor generating external input to DEVS Agent model
Metrics Visualization GUI
DEVS Agent
deltint("passive")
deltext("active"), processInfo()
DEVS/SOA or DEVSJAVA

Figure 5: DEVS wrapper implementation over an Axis web service client

A DEVS model has two modes of operation: an internal behavior representation and an external behavior representation. In developing a DEVS wrapper, which would be effectively a DEVS web service client, we will implement the external behavior. The concept is shown in the top half of Figure 5. The detail is shown in the lower half of the same Figure 5. It shows the mapping between the Axis layers, specifically the Axis binding layer and the DEVS elements. It describes the external event that is triggered whenever there is message exchange through the Axis client. This triggered event informs the DEVS atomic model that wraps this Axis client. Such an arrangement does not create any bottleneck or any pipe between the actual Axis client and the network. The DEVS wrapper is informed of the round-trip-time (RTT) when the actual service has been executed its completion. Consequently, it is a passive observer and offers no interference to the true communication between the client and the live web service. By inserting a specific set of code in any Axis generated client, we can create a DEVS wrapper that is ready to become a part of a test-agent federation coupled system, as described in Section 6. Further, having such automated design, it allows augmentation of a comprehensive log mechanism that can provide many other instrumentation data than just RTT.

Having described the basic DEVS Web service wrapper, the next task in line is the creation of a coupled model, a web service workflow to be more specific to actually utilize the DEVS modeling and simulation capabilities. The coupled model where this DEVS WSDL model is a component of a bigger networked model is not the focus of this article and more details will be available in the extended article. It is not hard to understand that once you have an atomic model, it can be easily used as a component in a DEVS coupled model.

5. IMPLEMENTATION OF DEVS AGENT

This case study demonstrates the execution of a web service encapsulated in a DEVS wrapper Agent and the associated obtained statistics.

5.1. Web Service Work Flow Formalism

We compose a process workflow based on certain goals, objectives or requirements. We can deduce the information we need to compose a workflow and develop an automated procedure towards DEVS based design and analysis. The information set for a Web Service workflow can be described in a four element tuple as:

WSWF: **< W,M,F,X>**
where,

W: Set of Web service definitions (WSDLs) or Agents each with a valid URL
M: Set of web service methods
F: defined as <C,L,D>
C is a set of W-M pairs with each pair as a source or destination
L is a set of partner links with each link containing a src and dest pair defined in C
D is a type of workflow mode which can either be a sequence, while, *holdSend* or *concurrent type*, which are corresponding to the BPEL specifications
X. Set of messages, where
Each Message contains Data and is defined by time of entry in system, rate, whether it is periodic or stochastic and can be either an Input message or an Output message

5.2. DEVS Wrapper Agent

In this most basic demonstration, we use only one web service. This web service executes a chat session between two users. The schematic is shown in Figure 6. In our example, we execute the session with a live person and a DEVS agent. The live person here is 'Jim Client' that connects to the CHAT service via an Internet browser at (CHAT, 2007). The chat session is executed using the GUI as shown in Figure 7.

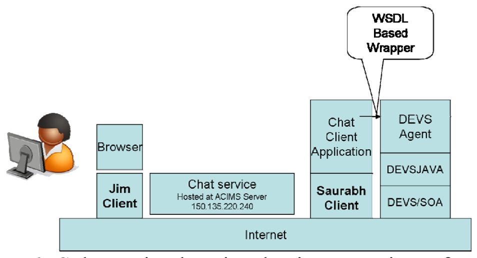
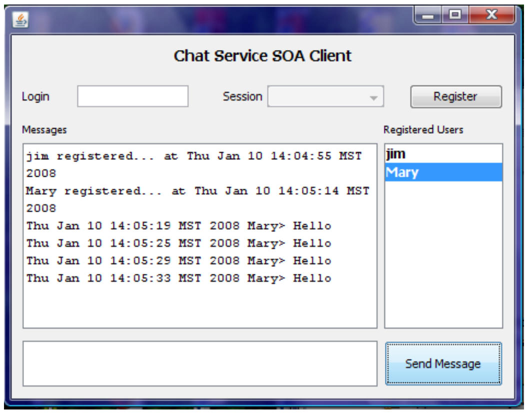


Figure 6: Schematic showing basic execution of DEVS Wrapper agent

The DEVS agent is defined according to the WSWF formalism as follows:

```
<W>: "CHAT":
    <W1:CHAT>:http://150.135.220.240:8080/C
hatServiceCollaboration/services/ChatServic
e?wsdl
    <A1:Jim>: "Jim:localhost:8080"
<M>: "Methods":
    <M1> postMessage()
    <M2> getAllMessages()
    <M3> getLastMessageId()
    <M4> registerAuthor()
    <M5> getUsers()
    <M6> getAllMessagesForAuthor()
<F>:"Flow specifications"
    <C>
<C1:Src>A1-M1
<C2:Src>A1-M2
<C3:Src>A1-M4
<C4:Src>A1-M5
<C5:Dest>W1-M1
<C6:Dest>W1-M2
<C7:Dest>W1-M4
<C8:Dest>W1-M5
    <L>
        <L1>C1-C5
        <L2>C2-C6
        <L3>C3-C7
        <L4>C4-C8
    <D>
        <D1>M1-HoldSend
        <D2>M2-While-infinity
        <D3>M4-HoldSend
        <D4>M5-While-infinity
<X>: Set of Messages
    <InputMsg>
        <I1>W1-M1{string:T1:0:false:false}
        <I2>W1-M4{string:T0:0.1:true:false}
    <OutputMsg>
        <O1>M2{string:T2:1:true:false}
        <O2>M5{string:T2:1:true:false}
```


Chat Service SOA Client

Login | Session | Register

Messages | Registered Users

```
jim registered... at Thu Jan 10 14:04:55 MST
2008
Mary registered... at Thu Jan 10 14:05:14 MST
2008
Thu Jan 10 14:05:19 MST 2008 Mary> Hello
Thu Jan 10 14:05:25 MST 2008 Mary> Hello
Thu Jan 10 14:05:29 MST 2008 Mary> Hello
Thu Jan 10 14:05:33 MST 2008 Mary> Hello
```

jim
Mary

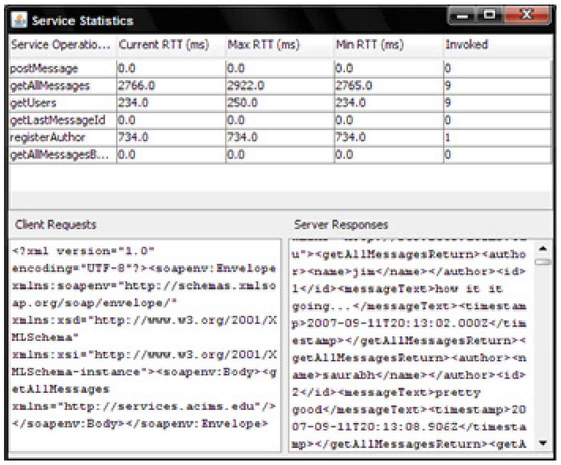
Send Message

Figure 7: Chat Service Client connected to CHAT Service

<W> tag contains description of the Chat Web Service as W1 and the agent description as A1 along with their URL. **<M>** contains the list of service methods that may be used in the process flow. **<F>** contains the flow description categorized into **<C,L,D>** as per the WSWF. **<C>** provides the source and destination specification for a W/A defined in *<W>* with *<M>*. **<L>** specifies the coupling between the sources and destinations as defined in *<C>*. **<D>** contains the execution of methods in *<M>* in logic implementation. For example, `<D1>M1-HoldSend` implies that the method *M1* is to executed in *HoldSend* manner. Similarly, `<D2>M2-While-infinity` implies that *M2* will be executed indefinitely when invoked or bounded by any condition. **<X>** specifies the input and output message structures in **<InputMsg,OutputMsg>** tags. The structure of **<InputMsg>** as specified in WSWF SES is *<SystemComponent-Method{Data: time of Start: R+: isPeriodic: isRandom>*. For example, the specification `<I1>W1-M1{string:T1:0:false:false}` implies that the message *I1* is an input to *W1*, method *M1* with data as *string*. It starts at *T1* with period 0. Any non-zero value means that the message will be incoming at a periodic rate. The next boolean variable '*false*' implies that it is not periodic. The last variable '*false*' implies that it is not random either. Similary, `<I2>W1-M4{string:T0:0.1:true:false}` implies that *M4* at *W1* is to be invoked by *string* data message with a periodic rate of 0.1. The <OutputMsg> has a similar structure except the fact that it does not contain any information about the system component. It only contains information about the method in <M> as it is just an output message. Whenever method <Mx> is invoked, it returns with the parametric details as in `<O1>M2{string:T2:1:true:false}`.

It is worth stressing here that the messages flow through the linkages as specified in **<L>**. This acts as a coupling for the DEVS models. There are two DEVS models in the WSWF instance described above, viz. *W1* and *A1*. Based on the coupling information for ex. `<L4>C4-C8` implies that the source is Agent `<C4:Src>A1-M5` and the destination is Web service `<C8:Dest>W1-M5`. The source sends a message invoking method *M5* at the destination. If there is a specification on how *M5* should be invoked in <*InputMsg*> listing, then the source has to ensure that it conforms to that specification. In this example there is no specification for M5. This implies that there are no parameters to be passed, but just the invocation. At the destination side, M5 has a specification `<O2>M5{string:T2:1:true:false}`, which implies that whenever *M5* returns a value, it will according to this <*OutputMsg*> specification.

The statistics for each of the methods in <*M*> is gathered according to the autogenerated agent GUI monitor at the agent's end. The statistics are largely the round trip time (RTT) for each of <*M*>. The GUI in Figure 8 also shows the SOAP messages that are exchanged between the pairs as specified in <*W*>.



Service Operatio...	Current RTT (ms)	Max RTT (ms)	Min RTT (ms)	Invoked
postMessage	0.0	0.0	0.0	0
getAllMessages	2766.0	2922.0	2765.0	9
getUsers	234.0	250.0	234.0	9
getLastMessageId	0.0	0.0	0.0	0
registerAuthor	734.0	734.0	734.0	1
getAllMessagesB...	0.0	0.0	0.0	0

Client Requests

```
<?xml version="1.0"
encoding="UTF-8"?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlso
ap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/X
MLSchema"
xmlns:xsi="http://www.w3.org/2001/X
MLSchema-instance"><soapenv:Body><g
etAllMessages
xmlns="http://services.acims.edu"/>
</soapenv:Body></soapenv:Envelope>
```

Server Responses

```
u"><getAllMessagesReturn><autho
r><name>jim</name></author><id>
1</id><messageText>how it it
going...</messageText><timestam
p>2007-09-11T20:13:02.000Z</tim
estamp></getAllMessagesReturn><
getAllMessagesReturn><author><n
ame>saurabh</name></author><id>
2</id><messageText>pretty
good</messageText><timestamp>20
07-09-11T20:13:08.906Z</timesta
mp></getAllMessagesReturn><getA
```

Figure 8: Associated Statistics GUI for an encapsulated Web Service in DEVS atomic model

6. MULTI-LAYERED AGENT BASED TEST INSTRUMENTATION SYSTEM USING GIG/SOA

A DEVS distributed federation is a DEVS coupled model whose components reside on different network nodes and whose coupling is implemented through middleware connectivity characteristic of the environment, e.g., SOAP for GIG/SOA, The federation models are executed by DEVS simulator nodes that provide the time and data exchange coordination as specified in the DEVS abstract simulator protocol. The DEVS Agent Monitoring System (TIS) is a DEVS coupled system that observes and evaluates the operation of the DEVS coupled system model. The DEVS models used to observe other participants are the DEVS test-agents. The TIS should provide a minimally intrusive test capability to support rigorous, on-going, repeatable and consistent testing and evaluation (T&E). Requirements for such a test implementation system include ability to

1. deploy agents to interface with SoS component systems in specified assignments
2. enable agents to exchange information and coordinate their behaviors to achieve specified experimental frame data processing
3. respond in real-time to queries for test results while testing is still in progress
4. provide real-time alerts when conditions are detected that would invalidate results or otherwise indicate that intervention is required
5. centrally collect and process test results on demand, periodically, and/or at termination of testing.
6. support consistent transfer and reuse of test cases/configurations from past test events to future test events, enabling life-cycle tracking of SoS performance.
7. enable rapid development of new test cases and configurations to keep up with the reduced SoS development times expected to characterize the reusable web service-based development supported on the GIG/SOA.

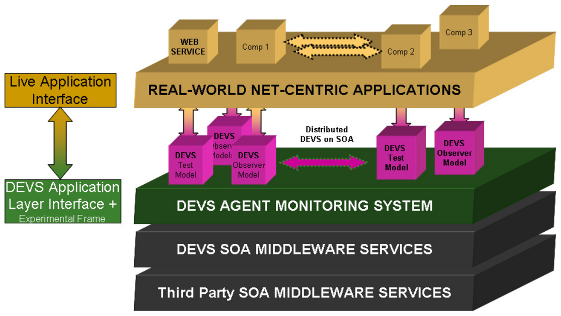
Many of these requirements are not achievable with current manually-based data collection and testing. Instrumentation and automation are needed to meet these requirements.

Net-centric Service Oriented Architecture (SOA) provides a currently relevant technologically feasible realization of the concept. As discussed earlier, the DEVS/SOA infrastructure enables DEVS models, and test agents in particular, to be deployed to the network nodes of interest. Details on how such observers can be auto-generated and be executed using DEVS/SOA are provided in (Mittal, Zeigler, Martin, Sahin and Jamshidi, 2008; Zeigler and Hammonds, 2007).

6.1. Deploying Test Agents over the GIG/SOA

Figure 9 depicts a logical formulation test federation that can observe a SUT to verify the message flow among components as derived from information exchange requirements. In this context, a mission thread is a series of activities executed by operational nodes. In playing out this thread, DEVS test models are informed of the current activities (or see to detect their onset) as well as the operational nodes that execute these messages. These test models watch messages sent and received by the components that host the participating operational nodes. The test models check whether such messages are the ones that should be sent or received under the current function.

The test-agents are contained in DEVS Experimental Frames (EF) are implemented as DEVS models, and distributed EFs are implemented as DEVS models, or agents as we have called them, reside on network nodes. Such a federation, illustrated in Figure 10, consists of DEVS simulators executing on web servers on the nodes exchanging messages and obeying time relationships under the rules contained within their hosted DEVS models. This DEVS Agent Monitoring System that contains DEVS models interacts with real world web services through the DEVS agents that were described earlier.


WEB SERVICE
Comp 1
Comp 2
Comp 3
Live Application Interface
REAL-WORLD NET-CENTRIC APPLICATIONS
DEVS Test Model
DEVS Observer Model
Distributed DEVS on SOA
DEVS Application Layer Interface + Experimental Frame
DEVS AGENT MONITORING SYSTEM
DEVS SOA MIDDLEWARE SERVICES
Third Party SOA MIDDLEWARE SERVICES
Figure 9: Multi-layered Agent-based test instrumentation framework

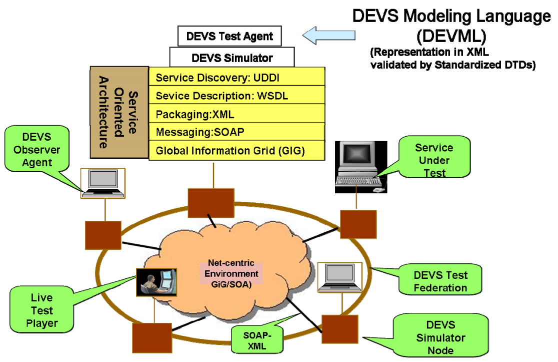
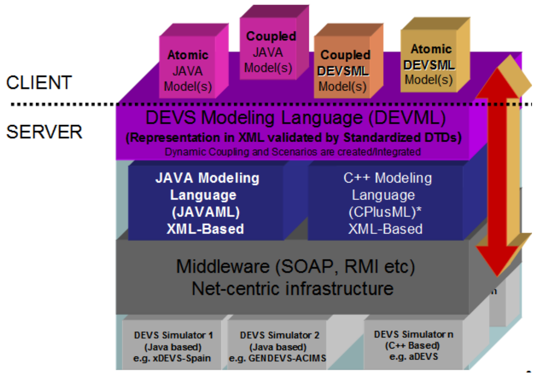


Figure 10: Prototypical DEVS Test Federation

6.2. Implementation of Test Federations

A test federation observes an orchestration of web-services to verify the message flow among participants adheres to information exchange requirements. As derived from requirement, a process workflow is a series of activities executed by operational nodes and employing the information processing functions of web-services. As discussed in (Mittal, Zeigler, Martin, Sahin and Jamshidi, 2008; Zeigler and Hammonds, 2007), test agents watch messages sent and received by the services that host the participating operational nodes. Depending on the mode of testing, the test architecture may, or may not, have knowledge of the driving process workflow under test. If it is available, DEVS test agents can be aware of the current activity of the operational nodes it is observing. This enables it to focus more efficiently on a smaller set of messages that are likely to provide test opportunities.

To help automate set-up of the test we use a capability to inter-covert between DEVS and XML. DEVSML (Mittal, Martin and Zeigler, 2007) allows distributing DEVS models in the form of XML documents to remote nodes where they can be coupled with local service components to compose a federation (Mittal, Martin and Zeigler 2007, Mittal, 2007) The layered middleware architecture capability is shown in 11 and (Mittal, Martin, and Zeigler 2007)


Atomic JAVA Model(s)
Coupled JAVA Model(s)
Coupled DEVSML Model(s)
Atomic DEVSML Model(s)
CLIENT
SERVER
DEVS Modeling Language (DEVML)
(Representation in XML validated by Standardized DTDs)
Dynamic Coupling and Scenarios are created/integrated
JAVA Modeling Language (JAVAML) XML-Based
C++ Modeling Language (CPlusML)* XML-Based
Middleware (SOAP, RMI etc) Net-centric infrastructure
DEVS Simulator 1 (Java based) e.g. xDEVS-Spain
DEVS Simulator 2 (Java based) e.g. GENDEVS-ACIMS
DEVS Simulator n (C++ Based) e.g. aDEVS
Figure 11: Layered Architecture of DEVSML towards transparent simulators in Net-centric domain

At the top is the application layer that contains model in DEVS/JAVA or DEVSML. The second layer is the DEVSML layer itself that provides seamless integration, composition and dynamic scenario construction resulting in portable models in DEVSML that are complete in every respect. These DEVSML models can be ported to any remote location using the web-service infrastructure and be executed at any remote location.

The simulation engine is totally transparent to model execution over the net-centric infrastructure. The DEVSML model description files in XML contains meta-data information about its compliance with various simulation 'builds' or versions to provide true interoperability between various simulator engine implementations. Such run-time interoperability provides great advantage when models from different repositories are used to compose models using DEVSML seamless integration capabilities. Recent articles provide an evidence in the direction to achieve interoperability for DEVS and non-DEVS models (Zeigler, Mittal and Hu, 2008; Mittal, Zeigler and Martin, 2008). Finally, the test federation is illustrated in Figure 10 where different models (federates) in DEVSML collaborate for a simulation exercise over GIG/SOA.

This section has laid out the framework on the creation and execution of a DEVS-based test instrumentation system.

7. CONCLUSIONS

Service Oriented Architecture (SOA) is still under development and many of the businesses are seriously considering migration of their IT systems towards SOAs. DoD's initiative towards migration of GIG/SOA and NCES requires reliability and robustness, not only in the execution but in the design and analysis phase as well. Web service orchestration is not just a research issue but a more practical issue for which there is dire need. Further, Service Oriented Architecture must be taken as another instance of system engineering for which there must be laid out engineering process. Modeling and Simulation provides the needed edge. Lack of methodologies to support design and analysis of such orchestration except BPEL related efforts cost millions in failure. This research has proposed that Discrete Event Formalism can be used to compose and analyze Web service workflows. The DEVS theory, which is based on system theoretic concepts, gives solid grounding in the modeling and simulation domain.

We have shown how a web service can be encapsulated into a DEVS atomic model and can be put towards a coupled DEVS system with other live web services as well as other DEVS models. We also have demonstrated the proposed use of Web Service Work Flow (WSWF) formalism in composing SOA, much like of the same functionalities of BPEL. We have also described creation of DEVS net-centric coupled systems based on SOA. We have also shown how the developed DEVS coupled system can be simulated using the basic DEVSJAVA framework as well as distributed DEVS/SOA framework. Further, on the basis of our earlier work on DEVS/SOA we have basis for:

- Agent-Implemented Test Instrumentation
- Net-centric Execution using Simulation Services
- Distributed Multi-level Test Federations
- Analysis that can help optimally tune the instrumentation to provide confident scalability predictions.
- Mission Thread testing and data gathering:
- Definition and implementation of military-relevant mission threads to enable constructing and/or validating models of user activity.
- Comparison with current commercial testing tools shows that by replicating such models in large numbers it will be possible to produce more reliable load models than offered by conventional use of scripts.

We have taken the challenge of constructing net-centric systems as one of designing an infrastructure to integrate existing Web services as components, each with its own structure and behavior with DEVS components and agents. The net-centric system is analogous to a System of System (SoS) where in hierarchical coupled models could be created. Various workflows can be integrated together using component based design. The net-centric system can be specified in many available frameworks such as BPMN/BPEL, UML, or by using an integrative systems engineering-based framework such as DEVS.

In this research, we illustrated how M&S can be used strategically to provide early feasibility studies and aid the design process. We have established the capability to develop a live workflow example with complete DEVS interface. In this role, DEVS acts as a full net-centric production environment. Being DEVS enabled, it is also executable as a system under test (SUT) model towards various verification and validation analysis that can be performed by coupling this SUT with other DEVS test models. Last but not the least, the developed DEVS system can be executed by both real and virtual users to the advantage of various performance and evaluation studies.

As components comprising SoS are designed and analyzed, their integration and communication is the most critical part that must be addressed by the employed SoS M&S framework. We discussed DoD's Global Information Grid (GIG) as providing an integration infrastructure for SoS in the context of constructing collaborations of web services using the Service Oriented Architecture (SOA). The present research is being considered and refined for testing GIG/SOA at Joint Interoperability Test Command [JITC], which is *the* agency to test future DoD systems. Clearly, the theory and methodology for such net-centric SoS development and testing are at their early stages.

REFERENCES

ACIMS software site, Available from http://www.acims.arizona.edu/SOFTWARE/software.shtml [Accessed July 2008]

IBM, BEA Systems, Microsoft, SAP AG, Siebel, 2007, Business Process Execution Language for Web Services version 1.1, Available from http://www.ibm.com/developerworks/library/specification/ws-bpel/ [Accessed July 2008]

Business Process Modeling Notation http://www.bpmn.org [Accessed July 2008]

Mittal, S., 2007, CHAT SOA web service, Available from http://www.saurabh-mittal.com/demos/ChatClient, [Accessed July 2008]

Mittal, S., 2007, DUNIP: A Prototype Demonstration, Available from http://www.acims.arizona.edu/dunip/dunip.avi [Accessed July 2008]

Joint Interoperability Test Command, a Defense Information Systems Agency at http://jitc.fhu.disa.mil/ [Accessed July 2008]

Department of Defense GIG Architectural Vision, Ver. 1.0, 2007, prepared by DoD CIO, available from http://www.defenselink.mil/cio-nii/docs/GIGArchVision.pdf [Accessed July 2008]

Hwang, M.H., Zeigler, B.P., 2006, A Modular Verification Framework using Finite and Deterministic DEVS, Proceedings of 2006 DEVS Symposium, pp. 57-65, April, Huntsville, Alabama, USA

Mittal, S., Martin, J.L., Zeigler, B.P., 2007, DEVS/SOA: A Cross Platform framework for Net-Centric Modeling and Simulation in DEVS Unified Process, SIMULATION: Transactions of SCS, under review

Mittal, S., Martin, J.L., Zeigler, B.P., 2007, DEVSML: Automating DEVS Simulation over SOA using Transparent Simulators", DEVS Symposium, March, Norfolk Virginia

Mittal, S., Martin, J.L., Zeigler, B.P., 2007, DEVS-Based Web Services for Net-centric T&E, Summer Computer Simulation Conference, San Diego, CA, USA

Mittal, S, 2007, DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures, Ph. D. Dissertation, Dept. of Electrical and Computer Engineering, University of Arizona

Mittal, S., Hwang, M.H., Zeigler, B.P, 2007, XFD-DEVS: An Implementation of W3C Schema for Finite Deterministic DEVS, Demo available at: http://www.saurabh-mittal.com/fddevs

Mittal, S., Zeigler, B.P., Martin, J.L.R., Sahin, F., Jamshidi, M., 2008, Modeling and Simulation for System of Systems Engineering, in Jamshidi, M., ed. Systems of Systems engineering for 21st Century, in press

Mittal, S., Martin, J.L.R., Zeigler, B.P., Nutaro, J.J., 2008, Design and Analysis of Service Oriented Architectures using DEVS/SOA-Based Modeling and Simulation, submitted to SIMULATION: Transactions of SCS

Mittal, S., Zeigler, B.P., Martin, J.L.R., 2008, Implementation of Formal Standard for Interoperability in M&S/System of Systems Integration with DEVS/SOA, Abstract accepted, manuscript in review at International Command and Control, C2 Journal

Net-Centric Enterprise Service http://www.disa.mil/nces/ [Accessed July 2008]

Pyke, Jon, 2007, XPDL: The Silent Workhorse of BPM, Available from http://www.bpm.com/FeatureRO.asp?FeatureId=232 [Accessed July 2008]

Mittal, S., 2007, DEVS/SOA sample demonstration in .avi format, Available from http://www.saurabh-mittal.com/demos/demoSOADEVS.avi [Accessed July 2008]

W3C Recommendation, 2007, Simple Object Access Protocol, Available from http://www.w3.org/TR/soap12-part1/ [Accessed July 2008]

W3C Recommendation, 2001, Web Services Description Language, Available from http://www.w3.org/TR/wsdl [Accessed July 2008]

Zeigler, B.P., Kim, T.G. and Praehofer, H., 2000, Theory of Modeling and Simulation. New York, NY, Academic Press.

Zeigler, B.P., Sarjoughian, H., DEVSJAVA, 2000: Available from http://www.acims.arizona.edu/SOFTWARE/devsjava_licensed/CBMSManuscript.zip [Accessed July 2008]

Zeigler, B.P., and Hammonds, P., 2007, Modeling & Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange, New York, NY: Academic Press.

Zeigler, B.P., Mittal, S., Hu, X., 2008, Towards a Formal Standard for Interoperability in M&S/Systems of Systems Engineering, Critical Issues in C4I, AFCEA-George Mason University Symposium, May, Washington, D.C, USA

AUTHORS BIOGRAPHY

Saurabh Mittal is the CEO at DUNIP Technologies, India. Previously he worked as Research Assistant Professor at the Department of Electrical and Computer Engineering at the University of Arizona where he received his Ph. D in 2007. His areas of interest include Web-based M&S using SOA, executable architectures, Distributed Simulation, and System of Systems engineering using DoDAF. He can be reached at saurabh.mittal@duniptechnologies.com

José L. Risco-Martín is an Assistant Professor in Complutense University of Madrid, Spain. He received his PhD from Complutense University of Madrid in 2004. His research interests are computational theory of modeling and simulation, with emphasis on DEVS, Dynamic memory management of embedded systems, and net-centric computing. He can be reached at jlrisco@dacya.ucm.es

Bernard P. Zeigler is Professor of Electrical and Computer Engineering at the University of Arizona, Tucson and Director of the Arizona Center for Integrative Modeling and Simulation. He is developing DEVS-methodology approaches for testing mission thread end-to-end interoperability and combat effectiveness of Defense Department acquisitions and transitions to the Global Information Grid with its Service Oriented Architecture (GIG/SOA). He can be reached at zeigler@ece.arizona.edu

Jesús M. de la Cruz is Professor at the Department of Computer Architecture and Automation at the Complutense University of Madrid, Spain, where he is the head of the Automatic Control and Robotics Group. His interest covers broad aspects of automatic control and its pplications, real time control, simulation, optimization, statistical learning, and robotics. He can be reached at jmcruz@fis.ucm.es.