DEVS-Based Simulation Web Services for Net-Centric T&E

Saurabh Mittal, José L. Risco*, Bernard P. Zeigler
{saurabh, zeigler}@ece.arizona.edu, *jlrisco@dacya.ucm.es

Arizona Center for Integrative M&S
ECE Department, University of Arizona
Tucson, AZ 85721

*Departamento de Arquitectura de Computadores
y Automática
Universidad Complutense de Madrid
28040 Madrid, Spain

Keywords:
DEVS, DEVSML, SOADEVS, Web services, XML, T&E

Abstract
Current test and evaluation (T&E) systems are not sufficiently well integrated with defined net-centric architectures to support system-of-systems and enterprise level testing. This paper discusses a test and development environment using Discrete Event System Specification Modeling Language (DEVSML) and the Service Oriented Architecture (SOA) framework. The underlying DEVS Modeling Language is built on XML and provides model interoperability among DEVS models hosted at remote network addresses. We describe the client application that communicates with multiple servers hosting DEVS Simulation services and the underlying SOADEVS framework. We show how SOADEVS is positioned to address the need for a DoD Architecture Framework, DoDAF-based net-centric paradigm for test and evaluation at the system-of-systems and enterprise systems levels. The SOADEVS framework provides the needed feature of run-time composability of coupled systems using the SOA framework.

1. Introduction
In an editorial [1], Carstairs asserts an acute need for a new testing paradigm that could provide answers to several challenges described in a three-tier structure. The lowest level, containing the individual systems or programs, does not present a problem. The second tier, consisting of systems of systems in which interoperability is critical, has not been addressed in a systematic manner. The third tier, the enterprise level, where joint and coalition operations are conducted, is even more problematic. Although current test and evaluation (T&E) systems are approaching adequacy for tier-two challenges, they are not sufficiently well integrated with defined architectures focusing on interoperability to meet those of tier three. To address mission thread testing at the second and third tiers, Carstairs advocates a collaborative distributed environment (CDE), which is a federation of new and existing facilities from commercial, military, and not-for-profit organizations. In such an environment, modeling and simulation (M&S) technologies can be exploited to support model-continuity [2] and model-driven design development [3], making test and evaluation an integral part of the design and operations life-cycle.

The development of such a distributed testing environment would have to comply with recent Department of Defense (DoD) mandates requiring that the DoD Architectural Framework (DoDAF) be adopted to express high-level system and operational requirements and architectures [4, 5, 6, 7]. Unfortunately, DoDAF and DoD net-centric [8] mandates pose significant challenges to testing and evaluation since DoDAF specifications must be evaluated to see if they meet requirements and objectives, yet they are not expressed in a form that is amenable to such evaluation. This Section begins by providing an overview of the current DEVS technology and the way in which DEVS is positioned to address the need for a DoDAF-based net-centric paradigm for test and evaluation at the system-of-systems and enterprise systems levels.

DEVS environments such as DEVSJAVA, DEVS-C++, and others [9] are embedded in object-oriented implementations, they support the goal of representing executable model architectures in an object-oriented representational language. As a mathematical formalism, DEVS is platform independent, and its implementations adhere to the DEVS protocol so that DEVS models easily translate from one form (e.g., C++) to another (e.g., Java) [10]. Moreover, DEVS environments, such as DEVSJAVA, execute on commercial, off-the-shelf desktops or workstations and employ state-of-the-art libraries to produce graphical output that complies with industry and international standards. DEVS environments are typically open architectures that have been extended to execute on various middleware such as the DoD's HLA standard, CORBA, SOAP, and others and can be readily interfaced to other engineering and simulation and modeling tools [2, 9, 27, 28, 30]. Furthermore, DEVS operation over web middleware (SOAP) enables it to fully participate in the net-centric environment of the Global Information Grid/ Service Oriented Architecture (GIG/SOA) [8]. As a result of recent advances, DEVS can support model continuity through a

simulation-based development and testing life cycle [2]. This means that the mapping of high-level requirement specifications into lower-level DEVS formalizations enables such specifications to be thoroughly tested in virtual simulation environments before being easily and consistently transitioned to operate in a real environment for further testing and fielding.

DEVS formalism categorically separates the Model, the Simulator and the Experimental frame. However, one of the major problems in this kind of mutually exclusively system is that the formalism implementation is itself limited by the underlying programming language. In other words, the model and the simulator exist in the same programming language. Consequently, legacy models as well as models that are available in one implementation are hard to translate from one language to another even though both the implementations are object oriented. Other constraints like libraries inherent in C++ and Java are another source of bottleneck that prevents such interoperability.

Brief Overview of Capabilities Provided by DEVS

To provide a brief overview of the current capabilities, Table 1 outlines how it could provide solutions to the challenges in net-centric design and evaluation. The net-centric DEVS framework requires enhancement to the basic DEVS capabilities, which are provided in later sections.

Desired M&S Capability for T&E	Solutions Provided by DEVS Technology
Support of DoDAF need for executable architectures using M&S such as mission based testing for GIG SOA	DEVS Unified Process [31] provides methodology and SOA infrastructure for integrated development and testing, extending DoDAF views [32].
Interoperability and cross-platform M&S using GIG/SOA	Simulation architecture is layered to accomplish the technology migration or run different technological scenarios [13, 17]. Provide net-centric composition and integration of DEVS 'validated' models using Simulation Web Services [19]
Automated test generation and deployment in distributed simulation	Separate a model from the act of simulation itself, which can be executed on single or multiple distributed platforms [10]. With its bifurcated test and development process, automated test generation is integral to this methodology [18].
Test artifact continuity and traceability through phases of system development	Provide rapid means of deployment using model-continuity principles and concepts like "simulation becomes the reality" [2].
Real time observation and control of test environment	Provide dynamic variable-structure component modeling to enable control and reconfiguration of simulation on the fly [14-17]. Provide dynamic simulation tuning, interoperability testing and benchmarking.

Table 1: Solutions provided by DEVS technology to support of M&S for T&E

The motivation for this work stems from this need of model interoperability between the disparate simulator implementations and provides a means to make the simulator transparent to model execution. We proposed DEVS Modeling Language (DEVSML) [19] that is built on eXtensible Markup Language (XML) [34] as the preferred means to provide such transparent simulator implementation.

Furthermore, this work aims to develop and evaluate distributed simulation using the web service technology. After the development of World Wide Web, many efforts in the distributed simulation field have been made for modeling, executing simulation and creating model libraries that can be assembled and executed over WWW. By means of XML and web services technology these efforts have entered upon a new phase. A prototype simulation framework has been implemented using web services technology. The central point resides in executing the simulator as a web service. The development of this kind of frameworks will help to solve large-scale problems and guarantees interoperability among different networked systems and specifically DEVS-validated models. Providing server side design is outside the scope of this paper. This paper focuses on the overall approach, and specifically the client that communicates with the server.

The paper is organized as follows. The next section provides information about the related work in distributed simulation and DEVS standardization efforts. Section 3 deals with our

earlier work on DEVSML and introduces the concept of SOADEVS. Section 4 provides basic information about the underlying technologies for the development of DEVSML SOA framework. Section 5 provides detailed look at the architecture of SOADEVS. Section 6 presents the SOADEVS client with an illustrated example. Finally, Section 7 provides conclusion and the ongoing work.

2. Related Work

There have been a lot of efforts in the area of distributed simulation using parallelized DEVS formalism. Issues like 'causal dependency' [10] and 'synchronization problem' [20] have been adequately dealt with solutions like: 1. restriction of global simulation clock until all the models are in sync, or 2. rolling back the simulation of the model that has resulted in the causality error. Our chosen method of web centric simulation does not address these problems as they fall in a different domain. In our proposed work, the simulation engine rests solely on the Server. Consequently, the coordinator and the model simulators are always in sync.

Most of the existing web-centric simulation efforts consist of the following components:

1. *the Application*: the top level coupled model with (optional) integrated visualization.
2. *Model partitioner*: Element that partitions the model into various smaller coupled models to be executed at a different remote location
3. *Model deployer*: Element that deployed the smaller partitioned models to different locations
4. *Model initializer*: Element that initializes the partitioned model and make it ready for simulation
5. *Model Simulator*: Element that coordinate with root coordinator about the execution of partitioned model execution.

The Model Simulator design is almost same in all of the implementation and is derived directly from parallel DEVS formalism [10]. There are however, different methods to implement the former four elements. DEVS/Grid [21] uses all the components above. DEVS/P2P [22] implements step 2 using hierarchical model partitioning based on cost-based metric. DEVS/RMI [30] has a configuring engine that integrates the functionality of step 1, 2 and 3 above. DEVS/Cluster [23] is a multi-threaded distributed DEVS simulator built on CORBA, which again, is focused towards development of simulation engine.

As stated earlier, the efforts have been in the area of using the parallel DEVS and implementing the simulator engine in the same language as that of the model.

These efforts are in no means similar to what we had proposed in our paper [19]. Our work is focused towards interoperability at the application level, specifically, at the model level and hiding the simulator engine as a whole. We are focused towards taking XML just as a communication middleware, as used in SOAP, for existing DEVS models, but not as complete solution in itself. We would like the user or designer to code the behavior in any of the programming languages and let the DEVSML SOA architecture be responsible to create a coupled model, integrating code in either of the languages and delivering us with an executable model that can be simulated. The user need not learn any new syntax, any new language; however, what he must use is the standardized version of P-DEVS implementation such as DEVSJAVA Version 3.0 [9] (maintained at www.acims.arizona.edu).

This kind of capability where the user can integrate his model from models stored in any web repository, whether it contained public models of legacy systems or proprietary standardized models will provide more benefit to the industry as well as to the user, thereby truly realizing the model-reuse paradigm.

In further sections we will provide details about the SOADEVS server and client, design of DEVS Simulator interface and standardized libraries that are used in our implementation.

3. Underlying Technologies

3.1 DEVS

DEVS formalism consists of models, the simulator and the Experimental Frame. We will focus our attention to the two types of models i.e. atomic and coupled models. The atomic model is the irreducible model definitions that specify the behavior for any modeled entity. The coupled model is the aggregation/composition of two or more atomic models connected by explicit couplings. The coupled model N can itself be a part of component in a larger coupled model system giving rise to a hierarchical DEVS model construction. Detailed descriptions about DEVS Simulator, Experimental Frame and of both atomic and coupled models can be found in [10].

3.2 Web Services and Interoperability using XML

Service oriented Architecture (SOA) framework is a framework consisting of various W3C standards, in which various computational components are made available as 'services' interacting in an automated manner towards achieving machine-to-machine interoperable interaction over the network. The interface is specified using Web Service Description language (WSDL) [25] 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 request a 'service' using SOAP message that is transmitted via

HTTP in 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 message in a SOAP wrapper.

3.3 DEVSML

DEVSML is a novel way of writing DEVS models in XML language. This DEVSML is built on JAVAML, which is in fact, XML implementation of JAVA. The current development effort of DEVSML takes its power from the underlying JAVAML that is needed to specify the 'behavior' logic of atomic and coupled models. The DEVSML models are transformable back'n forth to java and to DEVSML. It is an attempt to provide interoperability between various models and create dynamic scenarios.

The layered architecture of the said capability is shown in Figure 1. At the top is the application layer that contains model in DEVSJAVA 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 net-centric infrastructure and be executed at any remote location. Another major advantage of such capability is total simulator 'transparency'. 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. This has been achieved for at least two independent simulation engines as they have an underlying DEVS protocol to adhere to. This has been made possible with the implementation of a single atomic DTD and a single coupled DTD that validates the DEVSML descriptions generated from these two implementations. Such run-time interoperability provides great advantage when models from different repositories are used to compose bigger coupled models using DEVSML seamless integration capabilities.

4. DEVSML and SOADEVS

In Section 3.3 we described DEVSML as a means to develop net-centric collaborative models resulting in a composite XML portable file that can be executed by the validated DEVS simulator. In this section we will illustrate how the DEVSML architecture aides the distributed execution over net-centric platform thereby offering simulator transparency using Simulation Services.

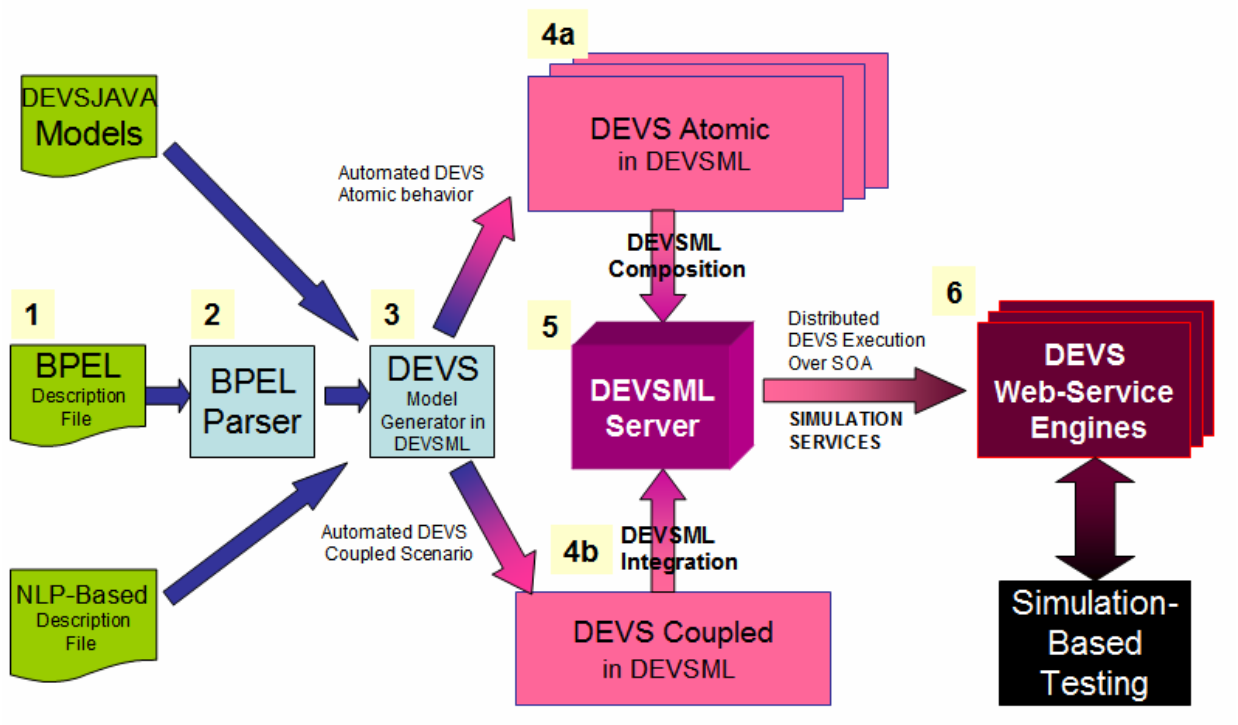

Figure 2: DEVSML and SOADEVS integrated

The DEVSML architecture is now divided in Client and Servers functionalities as shown below in Figure 1. The client provides model in DEVSJAVA or DEVSML, wherein they are transformable into each other and the Server end takes care of executing the simulation in a distributed manner using SOADEVS architecture.

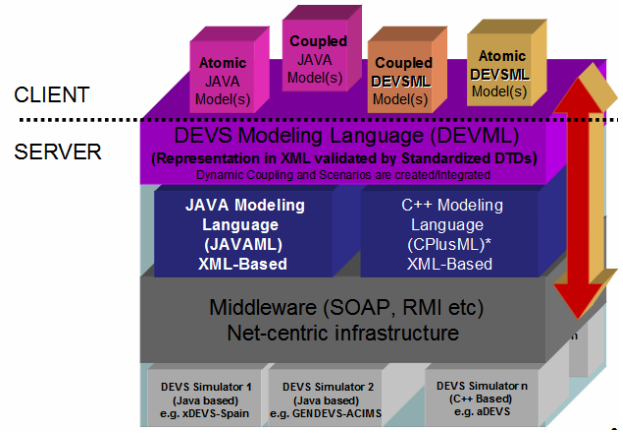

Figure 1: DEVSML implementation over SOADEVS.

Looking it in another perspective, the integration of DEVSML and SOADEVS is performed with the layout as shown in Figure 2. The manner in which DEVSJAVA models could be attained or developed by client can be manifold. The models can be created through Natural Language Processing (NLP) methods, raw .java format, or BPMN[1]/BPEL[2] files. Work is ongoing in the area of NLP and BPMN at ACIMS Center and will be reported in near future. The models rest with the client (Step 3, Figure 2). Once the client has DEVSJAVA models, DEVSML server can be used to integrate the client's model with model available at some other place on the web to get an enhanced integrated DEVSML file that can reproduce DEVSJAVA model in .java format (Step 4 and 5). The SOADEVS enabled server can either take this integrated DEVSML file directly or can ask user to provide the top-level coupled model through the SOADEVS client application. More details on this phase are provided in Section 6. Finally the remote simulation is conducted at various DEVS Simulation engines located over the web (Step 6) and be used for simulation-based testing in a distributed environment.

5. Distributed Simulation using SOADEVS

Web-based simulation requires the convergence of simulation methodology and WWW technology (mainly web service technology). 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. We are offering DEVS-based simulators as a web service, and they must have these 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).

Figure 3 shows the framework of the proposed distributed simulation using SOA. The complete setup requires one or more servers that are capable of running DEVS Simulation Service. The capability to run the simulation service is provided by the server side design of DEVS Simulation protocol supported by the latest DEVSJAVA Version 3.1.

The Simulation Service framework is two layered framework. The top-layer is the user coordination layer that oversees the lower layer. The lower layer is the true simulation service layer that executes the DEVS simulation protocol as a Service. The lower layer is transparent to the modeler and only the top-level is provided to the user. The top-level has four main services:

- Upload DEVS model
- Compile DEVS model
- Simulate DEVS model (centralized)
- Simulate DEVS model (distributed)

The second lower layer provides the DEVS Simulation protocol services:

- Initialize simulator i
- Run transition in simulator i
- Run lambda function in simulator i
- Inject message to simulator i
- Get time of next event from simulator i
- Get time advance from simulator i
- Get console log from all the simulators
- Finalize simulation service

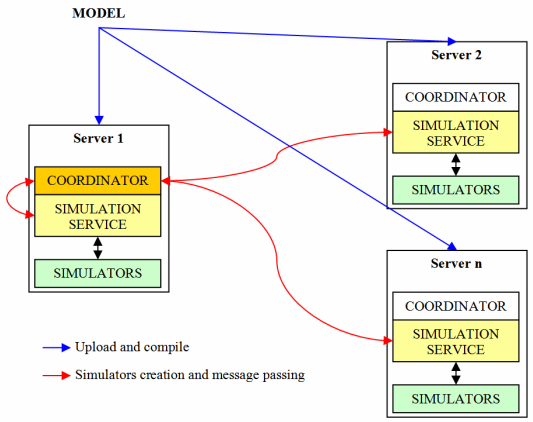

Figure 3: DEVS/SOA distributed architecture

[1] BPMN: Business Process Modeling Notation
[2] BPEL: Business Process Execution Language

The explicit transition functions, namely, the internal transition function, the external transition function, and the confluent transition function, are abstracted to a single transition function that is made available as a Service. The transition function that needs to be executed depends on the simulator implementation and is decided at the run-time. For example, if the simulator implements the Parallel DEVS (P-DEVS) formalism, it will choose among internal transition, external transition or confluent transition. Providing details about the abstracted transition function is outside the scope of this paper.

The client is provided a list of servers hosting DEVS Service. He selects some servers to distribute the simulation of his model. Then, the model is uploaded and compiled in all the servers. The main server selected creates a coordinator that creates simulators in the server where the coordinator resides and/or over the other servers selected.

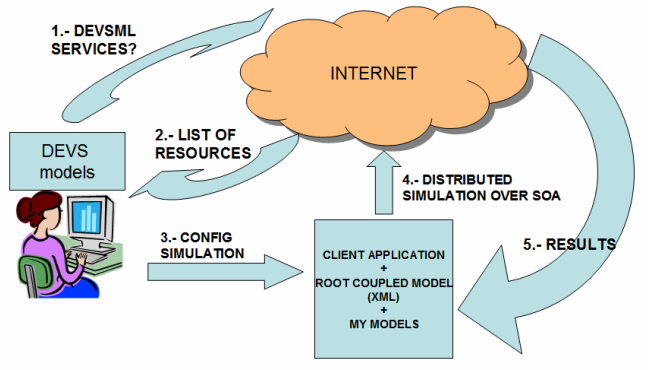

Figure 4: Execution of DEVS SOA-Based M&S

Summarizing from a user's perspective, the simulation process is done through three steps (Figure 4):

1. Write a DEVS model (currently DEVSJAVA is only supported).
2. Provide a list of DEVS servers (through UDDI, for example). Since we are testing the application, these services have not been published using UDDI by now. Select N number of servers from the list available.
3. Run the simulation (upload, compile and simulate) and wait for the results.

5.1 Abstraction of a Coupled Model with an Atomic Model with DEVS State Machine

One of the significant development steps we undertook in this effort is the masking of coupled model as an atomic model. Due to closure under coupling of the DEVS formalism we have an abstraction mechanism by which a coupled model can be executed like an atomic model. In contrast to the DEVS hierarchical modeling, where a coupled model is merely a container and has corresponding coupled-simulators (Figure 5), now it is considered an atomic model with lowest level atomic simulator (Figure 6). This has been accomplished by implementing an adapter as shown in Figure 6 above. The adapter Digraph2Atomic takes each coupled component of the model and uses it as an atomic model.

The number of simulators created depends on the number of components of the model at the top-level and the number of servers selected by the user. If the model contains 10 top-level components (including the contained digraphs) and the user select 5 servers, then 2 simulators are created in each server. After the whole simulation process, each simulation service sends a report back to the user containing information related to IP addresses and simulator assignment.

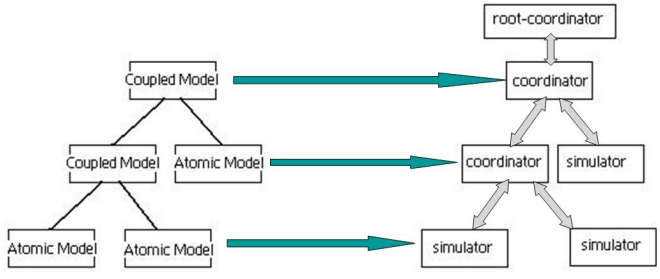

Figure 5: Hierarchical simulator assignment for a hierarchical model

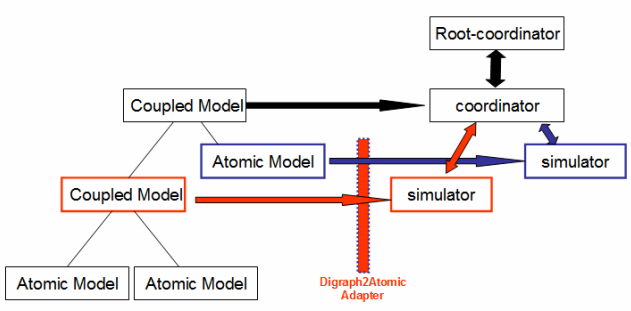

Figure 6: Hierarchical simulator assignment with Digraph2Atomic adapter

5.2. Message Serialization

The issue of message passing and models upload is done through serialization and SOA technologies. Figure 7 illustrates the message serialization process. When a component makes an external transition or executes the output function, the message received or emitted is serialized and then sent to the coordinator through the simulation service. The coordinator stores the location of each simulation service, so he is able to request all the messages after each iteration.

All the communication between the coordinator and simulation services is done through SOA protocol. The serialization is done through Java serialization utilities. In a newly developed real-time version, each simulator knows each simulation service at its end (from coupling information). So the communication can be solved by passing messages from simulation services to simulation services directly, without using the coordinator.

5.3 Centralized Simulation

The centralized simulation is done through a central coordinator which is located at the main server. The coordinator creates *n* simulation services over Internet. Each simulation service creates *m* simulators in order to simulation components of the model.

Figure 7 shows the process. Once the simulation starts, the coordinator executes the output function of the simulation services (in Figure 7: point 0 and 1). After that, the output is propagated and internal transitions occur. Propagating an output means that once the coordinator takes the serialized output from the simulation services (2 and 3), it is sent to other simulation services by means of coupling information (4 and 5). This information is known by the coordinator and no others as all messages must flow through the coordinator.

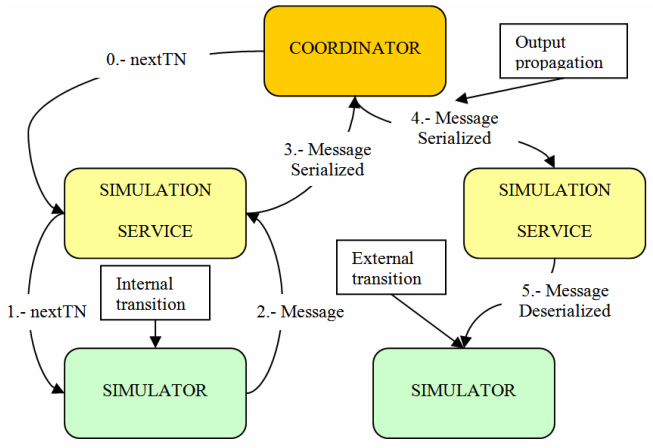

Figure 7: Centralized communication among services

As it appears, the coordinator participates in all message-passing and is the bottleneck. We designed distributed DEVS SOA protocol where the coupling information is downloaded to each of the models and coordinator is relieved of message-passing. It is described as follows.

5.4 Real-time Simulation

Real-time (RT) DEVS simulation is defined as the execution of DEVS simulation protocol in wall-clock time rather than logical time. For the real-time (RT) simulation we have incorporated one additional service to our SOA framework: the RT simulation service. This service extends the previous simulation service by means of two functions:

- Add external output function
- Start simulation

The design is similar in many aspects, but instead of a central coordinator, all the simulation is observed by an RT coordinator without any intervention. Furthermore, the RT simulation service creates RT simulators. Each RT simulation service knows the coupling information, so the message passing is made directly from simulation service to simulation service at the other end. The RT coordinator is located at the main server. This coordinator creates *n* RT simulation services over the Internet. Each simulation service creates *m* RT simulators in order to simulate the components of the model. After that the coupling information is broken down (on a per-model basis) and sent to the corresponding RT simulation service. Figure 8 illustrates the process. Once the simulation starts, the coordinator executes the *simulate* service and nothing else. The simulate service waits for internal or external transitions using real time (0). If an internal transition happens (1), the output is generated and propagated using the coupling information serializing and de-serializing messages (2,3 and 4).

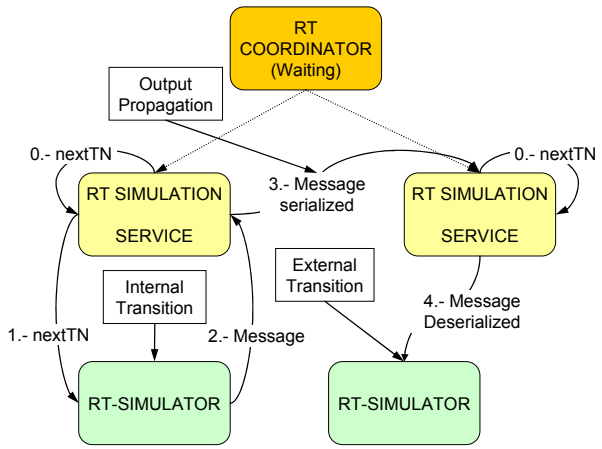

Figure 8: Real-time Communication among Services

5.5 Discussion

The difference between P-DEVS and classic DEVS is the handling of confluent function. The SOADEVS framework could have been built using other simulation formalisms. In fact, our simulation services could store any kind of simulator -as long as the service updates the simulation cycle according to the simulator engine selected. In the case of P-DEVS or DEVS, we have shown in Section 5, in the description of the simulation protocol services, that the service is independent in the sense of transition functions.

The user can freely consider both the centralized and distributed version of the simulation algorithm. This facility is provided at the second layer of services described in Section 5. However, the centralized mode performs much slower than the real-time distributed simulation due to obvious reasons of coordinator unloading. In development of SOADEVS client, we considered the real-time simulation as default option. Detailed performance analysis of both of these implementations is under progress and will be reported in our forthcoming publication.

6. SOADEVS Client

This Section provides the client application to execute DEVS model over an SOA framework using Simulation as a Service. From many-sided modes of DEVS model generation (Figure 2), the next step is the simulation of these models. The SOADEVS client takes the DEVS models package and through the dedicated servers hosting simulation services, it performs the following operations:

1. Upload the models to specific IP locations
2. Run-time compile at respective sites
3. Simulate the coupled-model
4. Receive the simulation output at client's end

The SOADEVS client as shown in Figure 9 operates in the following sequential manner:

1. The user selects the DEVS package folder at his machine
2. The top-level coupled model is selected as shown in Figure 9.
3. Various available servers are selected (Figure 10). Any number of available servers can be selected (one at least).
4. Clicking the button labelled "Assign Servers to Model Components" the user selects where is going to simulate each of the coupled models, including the top-level one, i.e., the main server where the coordinator will be created (Figure 10)
5. The user then uploads the model by clicking the Upload button. The models are partitioned and distributed among the servers chosen in the previous point
6. The user then compiles the models at the server's end by clicking the Compile button

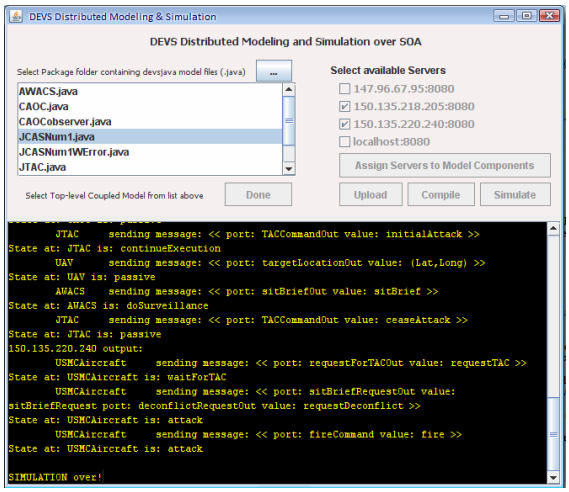

Figure 9: GUI snapshot of SOADEVS client hosting distributed simulation

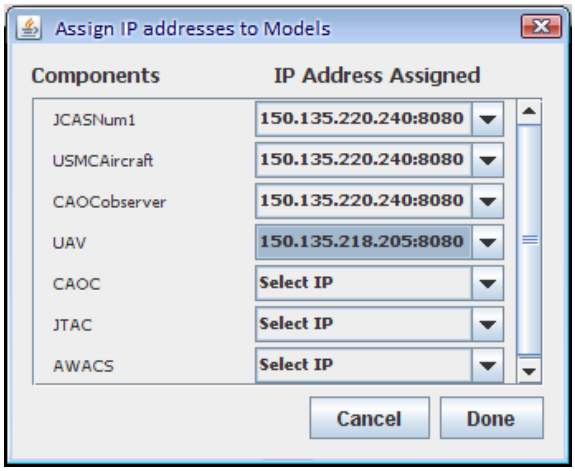

Figure 10: Server Assignment to Models

7. Conclusions and Future Work

We have addressed DEVSML as a medium towards composability and dynamic scenario construction. Furthermore, we have developed a Service Oriented Architecture framework for test and evaluation of DEVS models, called SOADEVS.

In this development effort, two implementations of DEVS simulation protocol have been presented. In the first, the simulation process is centralized by means of the Coordinator, which receives and propagates messages from one simulation service to others. There are no changes to the DEVS simulation protocol in this implementation but the

real-time Simulation service does require the simulation protocol to be tailored for SOA.

We also described the development of SOA client that provides DEVS-based Services specifically to execute the models as a running simulation. The primary 'simulation' service comprise of many helper services that were also developed. The server design or the back-end of SOADEVS is not the focus of this paper and will be reported in our forthcoming publication that will describe the WSDLs and their implementation.

This research work has presented proof of concept for DEVS based M&S over SOA. With the enhanced DoDAF [32], automated generation of DEVS model from DoDAF specifications can be executed and the architecture be simulated over a net-centric platform. The DUNIP [31] process also describes many other ways to autogenerate DEVS models from various other types of mission-thread specifications, for example, BPMN/BPEL and message-based restricted Natural Language Progressing (NLP). A Sample demonstration of DUNIP can be seen at [33]. In order to 'execute (as a model)' a set of scenario instructions over net-centric platform, the following capabilities must exist:

1. Transformation of the scenario specifications to a model, which is a DEVS model in this case
2. Execution of model over SOA
3. Communication using XML as middleware.

The first step is described in [31] and step 2 and 3 are presented in this paper. The next stage of analysis of this mission-thread statement is the development of automated test models and their execution over SOA. Automated test-model generation is discussed in [18, 31] and DEVS model execution can be performed by the work presented here.

Future Work

In terms of net-ready capability testing, what is required is the communication of live web services with those of test-models designed specifically for them. The approach we are working on has the following steps:

1. Specify the scenario
2. Develop the DEVS model
3. Develop the test-model from DEVS models
4. Run the model and test-model over SOA
5. Execute as a real-time simulation
6. Replace the model with actual web-service as intended in scenario.
7. Execute the test-models with real-world web services
8. Compare the results of steps 5 and 7.

Of course, there are many issues of policy management and security considerations that must be taken care of when test-models are communicating with live Web-Services. However, considering the fact that for any defense related mission-thread reliability testing the test-models would have the necessary security provisions, the 8-step process listed above can be executed. This work would also involve generation of DEVS models from WSDLs specifications. A small portion of BPMN-to-DEVS transformation is described in [31].

One other section that requires some description is the multi-platform simulation capability as provided by SOADEVS framework. It consists of realizing distributed simulation among different DEVS platforms or simulator engines such as DEVSJAVA, DEVS-C++, etc. In order to accomplish that, the simulation services will be developed that are focused on specific platforms, however, managed by a coordinator. In this manner, the whole model will be naturally partitioned according to their respective implementation platform and executing the native simulation service. This kind of interoperability where multi-platform simulations can be executed with our DEVSML integration facilities. DEVSML will be used to describe the whole hybrid model. At this level, the problem consists of message passing, which has been solved in this work by means of an adapter pattern in the design of the "message" class (used in Figures 7 and 8). Figure 11 shows a first approximation. The platform specific simulator generates messages or events, but the simulation services will transform these platform-specific-messages (PSMsg) to our current platform-independent-message (PIMsg) architecture developed in SOADEVS.

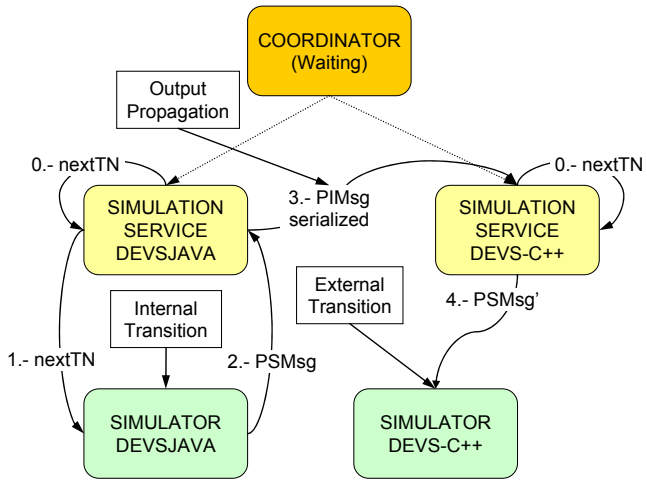

Figure 11: Future work

Hence, we see that the described SOADEVS framework can be extended towards net-ready capability testing. The SOADEVS framework also needs to be extended towards multi-platform simulation capabilities that allow test-models be written in any DEVS implementation (e.g. Java and C++) to interact with other as services.

References:

[1] D.J. Carstairs, *Wanted: A New Test Approach for Military Net-Centric Operations*, Guest Editorial, ITEA Journal, Volume 26, Number 3, October 2005

[2] X. Hu, and B.P. Zeigler, *Model Continuity in the Design of Dynamic Distributed Real-Time Systems*, IEEE Transactions on Systems, Man And Cybernetics— Part A, Volume 35, Issue 6, pp. 867-878, November 2005

[3] A. Wegmann, *Strengthening MDA by Drawing from the Living Systems Theory*, Workshop in Software Model Engineering, 2002

[4] DoD Architecture Framework, Software Productivity Consortium, http://www.software.org/pub/architecture/dodaf.asp, last accessed Jan 9, 2005.

[5] DOD Instruction 5000.2 *Operation of the Defense Acquisition System*, 12 May 2003.

[6] Chairman, JCS Instruction 3170.01D *Joint Capabilities Integration and Development System*, 12 March 2004.

[7] Chairman, JCS Instruction 6212.01C *Interoperability and Supportability of Information Technology and National Security Systems*, 20 November 2003

[8] K. Atkinson, *Modeling and Simulation Foundation for Capabilities Based Planning*, Simulation Interoperability Workshop Spring 2004

[9] ACIMS software site:
http://www.acims.arizona.edu/SOFTWARE/software.shtml

[10] B. P Zeigler, H. Praehofer, T. G. Kim, *Theory of Modeling and Simulation*, Academic Press, 2000

[11] *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies*. Editors: Hessam Sarjoughian, François E. Cellier. Spring-Verlag, Berlin, 2001.

[12] B. P.Zeigler, DEVS Today: Recent Advances in Discrete Event-based Information Technology, MASCOTS Conference, 2003

[13] H. Sarjoughian, B. Zeigler, and S. Hall, *A Layered Modeling and Simulation Architecture for Agent-Based System Development*, Proceedings of the IEEE 89 (2); 201-213, 2001

[14] S. Mittal, B. P. Zeigler, "Dynamic Simulation Control with Queue Visualization", Summer Computer Simulation Conference, SCSC'05, Philadelphia, July 2005

[15] S. Mittal, B. P. Zeigler, P. Hammonds, M. Veena, "Network Simulation Environment for Evaluation and Benchmarking HLA/RTI Experiments", JITC Report, Fort Huachuca, December 2004.

[16] X. Hu, B.P. Zeigler, S. Mittal, "Dynamic Configuration in DEVS Component-based Modeling and Simulation", SIMULATION: Transactions of the Society of Modeling and Simulation International, November 2003

[17] S. Mittal, B.P. Zeigler, "Modeling/Simulation Architecture for Autonomous Computing", Autonomic Computing Workshop: The Next Era of Computing, Tucson, January 2003.

[18] B.P. Zeigler, D. Fulton, P. Hammonds, J. Nutaro, "Framework for M&S Based System Development and Testing in Net-centric Environment", ITEA Journal, Vol. 26, No. 3, October 2005

[19] S. Mittal, J.L. Risco. *DEVSML: Automating DEVS Execution over SOA Towards Transparent Simulators*. In Special Session on DEVS Collaborative Execution and Systems Modeling over SOA, DEVS Integrative M&S Symposium DEVS'07, March 2007.

[20] R. M.. Fujimoto, *Parallel and Distribution Simulation Systems*, Wiley, 1999

[21] C. Seo, S. Park, B. Kim, S. Cheon, B.P. Zeigler, Implementation of Distributed High-performance DEVS Simulation Framework in the Grid Computing Environment, Advanced Simulation Technologies conference (ASTC), Arlington, VA, 2004

[22] S. Cheon, C. Seo, S. Park, B.P. Zeigler,, Design and Implementation of Distributed DEVS Simulation in a Peer to Peer Networked System, Advanced Simulation Technologies Conference, Arlington, VA, 2004

[23] K. Kim, W. Kang, CORBA-Based, Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-hierarchical One, International Conference on Computational Science and Its Applications, ICCSA, Italy 2004

[24] H. Vangheluwe, L. Bolduc, E. Posse, DEVS Standardization: some thoughts, Winter Simulation Conference 2001

[25] WSDL http://www.w3.org/TR/wsdl

[26] H. Sarjoughian, B.P. Zeigler, "DEVS and HLA: Complimentary Paradigms for M&S?" Transactions of the SCS, (17), 4, pp. 187-197, 2000

[27] Y. Cho, B.P. Zeigler, H. Sarjoughian, Design and Implementation of Distributed Real-Time DEVS/CORBA, IEEE Sys. Man. Cyber. Conf., Tucson, Oct. 2001.

[28] G. Wainer, N. Giambiasi, Timed Cell-DEVS: modeling and simulation of cell-spaces". Invited paper for the book Discrete Event Modeling & Simulation: Enabling Future Technologies, Springer-Verlag 2001

[29] XML: http://www.w3.org/XML/

[30] M. Zhang, B.P. Zeigler, P. Hammonds, DEVS/RMI-An Auto-Adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies, ITEA Journal, July 2005

[31] S. Mittal, "DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures", PhD Dissertation, University of Arizona, 2007

[32] S. Mittal, "Extending DoDAF to Allow DEVS-Based Modeling and Simulation", Special Issue on DoDAF, Journal of Defense Modeling and Simulation, Vol III, No. 2, 2006

[33] DUNIP: A Prototype demonstration:
http://www.acims.arizona.edu/dunip/dunip.avi

Biography

Saurabh Mittal

He is currently working as a research engineer at ACIMS lab, University of Arizona. He received his PhD in Computer Engineering from University of Arizona in 2007. His interests include DEVS Modeling theory, DoDAF applications, Net-centric computing and SOA. He can be reached at saumitt@gmail.com

José L. Risco-Martin

He is an assistant professor in Universidad Complutense de Madrid. 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@gmail.com

Bernard P. Zeigler

He is a professor at Electrical and Computer Engineering department at University of Arizona. He is an IEEE fellow and is best known for his formulation of DEVS theory. He can be reached at zeigler@ece.arizona.edu