Interoperability between DEVS and non-DEVS models using DEVS/SOA

José L. Risco-Martín[1], Alejandro Moreno[2], J. M. Cruz[1] and Joaquín Aranda[2]

[1]**Departamento de Arquitectura de Computadores y Automática**
Facultad de Informática
Universidad Complutense de Madrid (UCM)
28040 Madrid, Spain
{jlrisco,jmcruz}@dacya.ucm.es

[2]**Departamento de Informática y Automática**
Escuela Técnica Superior de Informatica
Universidad Nacional de Educación a Distancia (UNED)
28040 Madrid, Spain
amoreno@bec.uned.es, jaranda@dia.uned.es

Keywords: DEVS, Matlab, Standard, Interoperability, DEVS/SOA

Abstract

The absence of standarization could delay the progress of novel technological advances. Whereas, the guidance of uniform engineering specifications aims to amplify interoperability uppon systems. Extending functionality among different modeling and simulation frameworks opens a broad outreach to researchers. In this paper, we propose an approach supporting interoperability among DEVS compilant models and Matlab functionality in consort with a multi-platform web-based distributed simulation framework handling The *Service oriented Architecture (SOA)*. To prevent the lack of homogeneous criteria, we employ recently developed interoperability concepts accompanied by *Discrete EVent System (DEVS)* Specification formalism to implement a standard for interoperability: a distributed simulation environment involving the DEVS/SOA JAVA and .NET simulation engines to simulate DEVS and non-DEVS models. Furthermore, we present a distributed sample model following the DEVS models specification interface while integrating Matlab utilities.

1. INTRODUCTION

Interoperability is a quality that denotes the ability of diverse independent systems to work together [1]. If two or more organisms are capable of comunicating and exchanging data between themselves, the overall system is interoperable. Achieving a high degree of inteoperability in simulation is meant to be a primer objective uppon the research community [2]. The main reason is to assist the confluence between the large variety of legacy simulation frameworks and the rapidly developed modern simulations.

There are several current DEVS implementations providing interoperability employed at different levels. The DEVS/CORBA [3] distributed simulation environment offers an alternative implementation of discrete event system specification (DEVS) modeling and simulation theory based on CORBA communication middleware. Moreover, the DEVS/HLA [4] simulation environment supports high level model building using DEVS methodology as well as supplying heterogeneous simulation models interoperability based on HLA concepts. HLA is a network middleware layer that supports message exchanges among simulation components, called federates, in a neutral format and also provides a range of services to support dynamic and efficient execution of simulations. However, experience with HLA has been disappointing and forced proponents to acknowledge the difference between enabling heterogeneous simulations to exchange data, so-called technical interoperability, and substantive interoperability - the desired outcome of exchanging meaningful data so that coherent interaction among federates takes place [5]. Tolk introduced the *Levels of Conceptual Interoperability Model (LCIM)* which identified seven levels of interoperability among participating systems [6]. These levels also can be viewed as a refinement of the operational interoperability type which is one of three defined by Dimario [7]. The operational type concerns linkages between systems in their interactions with one another, the environment, and with users. The additional levels provide more elaboration to the catch-all category of substantive interoperability and, are missing from HLA standard as such.

A similar DEVS interoperability conceptualization is the work presented by Wuztler [8]. An approach that supports simultaneous execution of a set of DEVS-based models written in different programming languages throughout a definition of an abstract model. In spite of this similarity, our approach is encouraged by the newly approved DEVS interoperability standarization guidelines.

Recently, within a working group of the Simulation Inter-

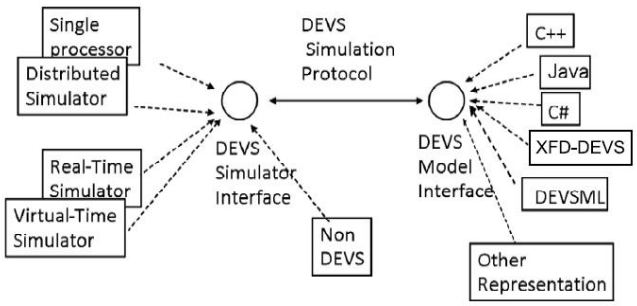

Figure 1. Conceptual Architecture of Standard

operability Standards Organization, a standard has been under development to support interoperability of DEVS models implemented in different platforms as well as with legacy simulations. Figure 1 illustrates an architectural approach proposed to accommodate the various combinations and permutations of possible application, both currently known, as well as those that will emerge in the future. The basic idea is to define two sets of interfaces; the DEVS model Interface and the DEVS Simulator Interface, as well as a DEVS Simulation Protocol that operates between the two. The interfaces protocols are based on those in GenDEVS, an implementation at the heart of the DEVJAVA M&S environment [9]. DEVS/C++, DEVSJAVA and xDEVS are platform specific implementations while DEVSML [10] and XFD-DEVS [11] are platform independent implementations in XML which can transform to any platform specific implementations. As a direct consequence of the model-simulator separation there can be multiple ways in which the same model can be simulated - all adhering to the abstract simulator specification. From the above introduction, we can infer that the standard will have multiple simulation scenarios.

In this work, we describe a Discrete Event System Specification (DEVS) simulation interoperability among two different DEVS Model implementations. A DEVS-compilant model written in Java executed along with a model extended by Matlab functionality. In addition, we succesfully demonstrate the application of DEVS/SOA net-centric modeling and simulation environment that uses XML-based SOA as frontend for DEVS simulator interoperation. Our proposal certifies the formerly stated conceptual architecture of standard approved by the Simulation Interoperability Standards Organization. The interoperability experiment basis insures both sides of Figure 1.

2. BACKGROUND

2.1. DEVS

DEVS formalism consists of models, the simulator and the experimental frame. We will focus our attention to the specified two types of models i.e. atomic and coupled models. The atomic model is the irreducible model definition that specifies the behavior for any modeled entity. The coupled model is the aggregation/composition of two or more atomic and coupled models connected by explicit couplings. The formal definition of parallel DEVS is given in [12]. An atomic model is defined by the following equation:

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

where,

- X is the set of input values
- S is the state space
- Y is the set of output values
- $\delta_{int} : S \rightarrow S$ is the internal transition function
- $\delta_{ext} : Q \times X^b \rightarrow S$ is the external transition function
  - $Q = \{(s,e) : s \in S, 0 \leq e \leq ta(s)\}$ is the total state set, where e is the time elapsed since last transition
  - X^b is a set of bags over elements in X
- δ_{con} is the confluent transition function, subject to $\delta_{con}(s,\emptyset) = \delta_{int}(s)$
- $\lambda : S \rightarrow Y$ is the output function
- $ta(s) : S \rightarrow \Re_0^+ \cup \infty$ is the time advance function.

The formal definition of a coupled model is described as:

$$N = \langle X, Y, D, EIC, EOC, IC \rangle \quad (2)$$

where,

- X is the set of external input events
- Y is the set of output events
- D is a set of DEVS component models
- EIC is the external input coupling relation
- EOC is the external output coupling relation
- IC is the internal coupling relation.

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.

Figure 2 shows a coupled DEVS model. M1 and M2 are DEVS models. M1 has two input ports: “in1” and “in2”, and one output port: “out”. The M2 has one input port: “in1”,

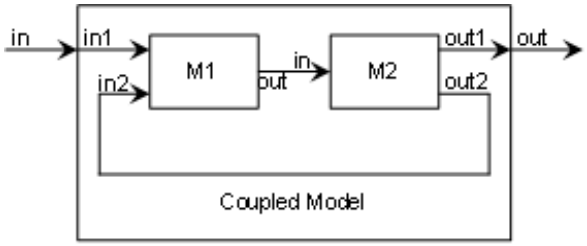

Figure 2. Coupled DEVS model

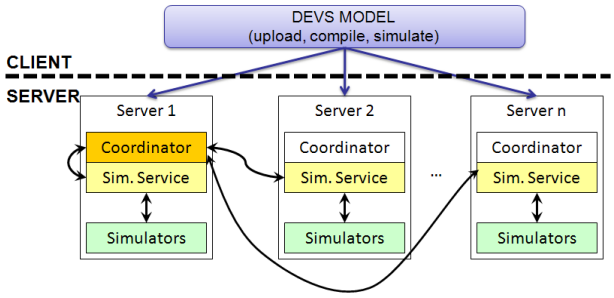

Figure 3. DEVS/SOA distributed architecture

and two output ports: "out1" and "out2". They are connected by input and output ports internally (this is the set of internal couplings, IC). M1 is connected by external input "in" of Coupled Model to "in1" port, which is an external input coupling (EIC). Finally, M2 is connected to output port "out" of Coupled Model, which is an external output coupling (EOC).

There are varied libraries for expressing DEVS models across the globe, such as DEVSJAVA [9], DEVS/C++ [9], CD++ [13], xDEVS [14], etc., and all of them have efficient implementations for executing the DEVS protocol. Providing the advantages of Object Oriented frameworks such as encapsulation, inheritance, and polymorphism. Plus, they all manage the simulation time, coordinates event schedules, and supply a library for simulation, a graphical user interface to view the results, and other utilities.

Detailed descriptions about DEVS Simulator, Experimental Frame and of both atomic and coupled models can be found in [12].

2.2. DEVS/SOA

The Service oriented Architecture (SOA) is a framework consisting of various W3C standards, in which various computational components are made available as "services" interacting in an automated manner achieve machine-to-machine interoperable interaction over the network. 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, which are based on 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 our 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.

Currently, there exists two implementations of DEVS/SOA. One for Java DEVS-based models (DEVS/SOA JAVA) and another for .NET DEVS-based model (DEVS/SOA .NET) [15].

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 three main services: upload DEVS model, compile DEVS model, and simulate DEVS model. 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, and finalize simulation service. 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 runtime. For example, if the simulator implements the *Parallel DEVS* formalism, it will choose among internal transition, external transition or confluent transition.

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. This whole framework is known as DEVS/SOA framework and details are available at [16], [17].

Summarizing from a user's perspective, the simulation process is done through three steps (Figure 4): (1) write a DEVS model (actually DEVSJAVA, xDEVS and DEVS.NET are supported), (2) provide a list of DEVS servers. Select *N* number of servers from the list available, and (3), run the simulation (upload, compile and simulate) and wait for the results.

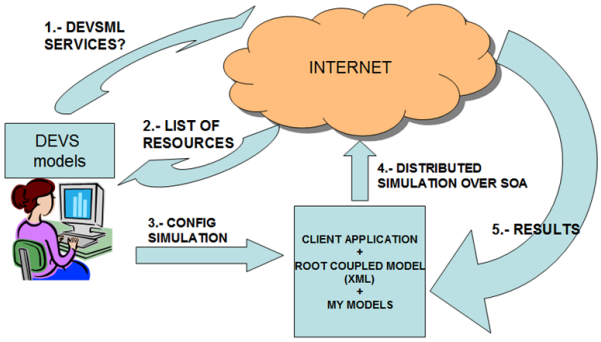

Figure 4. Execution of DEVS SOA-Based M&S

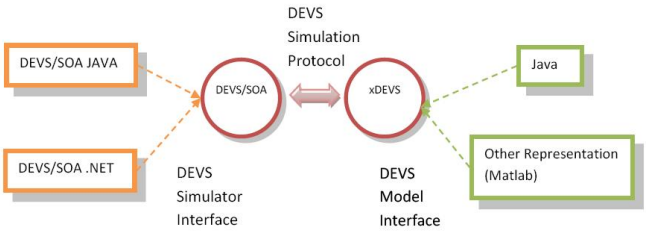

Figure 5. DEVS & Matlab Interoperability Conceptual Architecture of Standard

2.3. Matlab

Matlab is a numerical computing environment and programming language [18]. Created by The MathWorks, Matlab allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages.

Yet, for a long time Matlab was criticized since the software is a proprietary product of The MathWorks, users are subject to vendor product and services substantial switching costs. However, lately an additional tool called the Matlab Builder under the Application Deployment tools section has been provided to deploy Matlab functions as library files which can be used with .NET or Java application building environment.

Matlab *Java Builder (JA)* allows to integrate Matlab applications into Java programs by creating Matlab based Java classes that can be deployed royalty-free on desktop machines or Web servers. The Matlab Builder JA product creates the Java classes by encrypting Matlab functions and generating a Java wrapper around them.

Nevertheless, the drawback is that the computer where the application has to be deployed needs *Matlab Component Runtime (MCR)* for the Matlab files to function normally. Contrariwise MCR can be distributed freely with library files generated by the Matlab compiler.

3. DEVS INTEROPERABILITY

3.1. Introduction

Within the purpose of certifiyng the interoperability standard of cross-platform DEVS models aboard our approach, we refer to the *semantic level* interoperability [19]. As shown in Figure 5 we concentrate our efforts to satisfy the interoperation standards architecture involving different DEVS and non-DEVS models implementations, as well as, cooperation and information exchange among distributed simulators that differ on the computation principles.

Right side of Figure 5 suggests interoperation between a DEVS model implementation developed with Java programming language and non-DEVS functionality that benefits from functions encoded on Matlab programming language. Interoperability across distinct DEVS models achieved by the inheritance of the DEVS model interface xDEVS.

Moreover, Figure 5 illustrates standard distributed simulation among different DEVS simulation engines supported by our DEVS/SOA interface. A multi-platform simulation that comprehends a Java sustained simulation environment toghether with a .NET based simulation framework. Since both simulation platforms are based on web services using WSDL specification, they can be considered platform-independent simulation engines, i.e., a standard communication between both simulation platforms can be performed without any kind of middleware.

3.2. Modeling approach

In order to carry out our approach we take advantage of existing DEVS model implementation interface xDEVS [14], a DEVS *Application Programming Interface (API)* written in Java. To support our implementation modeling requirements, the Java API xDEVS specifies an interface for both atomic and coupled models. Although rigorously speaking, there is not a formal standard DEVS model interface, we are adopting the xDEVS interface as a point of union for two different simulation platforms (see Table 1).

Figure 6 depicts our integration methodology. In the first place we have DEVS model interface, naturallly implemented as an xDEVS interface. Keeping track of Figure 6 inheritance path, we slide down from DEVS model interface xDEVS to DEVS models implementations which ensure DEVS formalism especifications. These DEVS models can be implemented as full DEVS models or otherwise act as an adapter for Matlab integration. Further on, we explain in more detail Matlab integration tecnical requirements, now we focus on Matlab functionality abstraction.

Figure 6 illustrates Matlab Java wrapper function assembly process with Matlab Builder support. The application tool Matlab Java Builder allows the user to deploy Matlab functions by creating Matlab based Java classes which can be used with Java application building environment. This additional Matlab tool creates the Java classes by encrypting Matlab

Table 1. xDEVS interface of a DEVS model

```
public interface Devs {
        //DEVS time advance function.
        double ta();
        //DEVS internal transition function.
        void deltint();
        //DEVS external transition function.
        void deltext(double e, Message x);
        //DEVS confluent function.
        void deltcon(double e, Message x);
        //DEVS output function.
        Message lambda();
}
```

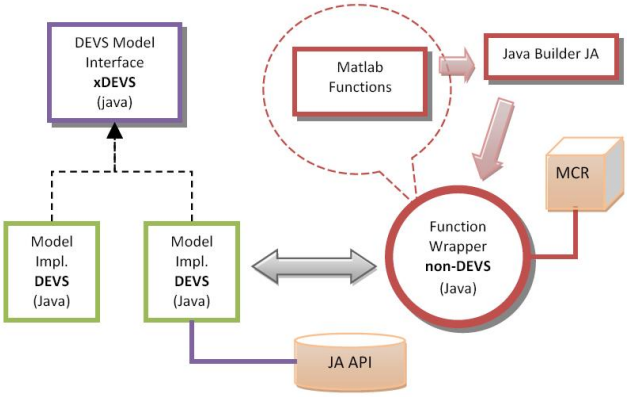

Figure 6. Implementation Overview

functions and generates a Java wrapper around them. The system introduces Matlab computation capabilities to our DEVS model adapter as non-DEVS functionality. The DEVS model implementation playing as a Matlab connector requests the inclusion of the encrypted Matlab functions translated as java classes in order to take advantage of their computing capacity. In addition, Figure 6 points out the major handicap concerning to the system operability, the machine where the application has to be deployed needs Matlab Component Runtime (MCR) for the Matlab files to function normally.

Now, we take a quick look at the tecnical procedures for Matlab integration. To enable Java applications to exchange data with Matlab methods they invoke, Java Builder provides an API. This package provides a set of data conversion classes. Each class represents a Matlab data type. As seen on Figure 6 Matlab integration within the DEVS model adapter requires the linkage to the Java Builder API. This API assists the information exchange process management among a Java application environment and Matlab programming framework.

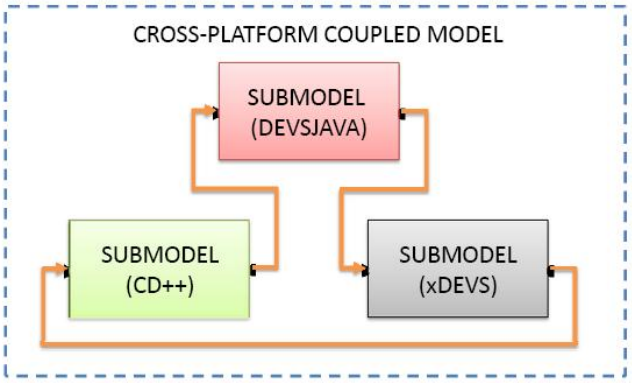

Figure 7. Cross Platform Coupled Model

Summarizing from a global perspective, we developed a DEVS model component within xDEVS interface global guidelines fulfilled with a Java model implementation. Alternatively, exploiting Matlab numerical computing environment we generated a wrapper of the desired Matlab funcionality managed by the specified model interface implementation.

Via the DEVS Simulator environment incorporated in the xDevs framework, we are capable of modeling and simulate atomic and coupled models that share the same semantics given the DEVS mathematical specification, but differ in the computing environment basis. As we have seen so far, implementations of both Java compilant models and Matlab core based models that share a common DEVS interface have been presented. Now, the DEVS Simulator xDevs that remains unchanged is able to simulate the formerly stated interoperable DEVS models implementations.

3.3. Simulation approach

The above methodology can be extended for DEVS/SOA simulation framework (left side of Figure 5). As explained above, DEVS/SOA environment provides DEVS-based simulators as a web service, which are based on standard communication technologies. Each atomic or coupled component may be implemented using different simulation engines, called platforms. Figure 7 depicts an example of multiplatform DEVS model. Even though the standard interface of interoperability for DEVS simulator engines does not cope with a formal specification, we assume the distributed DEVS/SOA framework architecture as the railway junction among two different simulation platforms.

Next, we will describe the combination of our extended DEVS model in addition with DEVS/SOA distributed simulation context. As soon as we build up the xDEVS based model with Matlab non-DEVS functionaility aforementioned. As illustrated on Figure 4, by means of DEVS/SOA framework, we request the list of servers hosting DEVS service. Then,

we select the servers to distribute the simulation of the whole model. Next, the entire model is uploaded and compiled in the chosen servers. The server selected as the leader creates the coordinator that in turn creates simulators in the servers where the coordinator resides over the other servers selected. The rest of the behavior of the application is the same that in our previous architecture. Messages are passed by means of an adapter pattern, they may be translated into different platforms. However, an outcome that affects DEVS/SOA current architecture heterogeneousness and usability is that the user must send the entire multi-platform model implementation. An efficient solution is proposed on the yet unpublished paper [17] based on a slight modification of the coordinator creation process. The structure description language DEVSML is applied to determine the root coupled model. This DEVSML document contains the location of each submodel and the main server. The document is transfered to the main server which will distribute the sub-models among its corresponding servers, and the simulation begins. On the other hand, again, the mayor drawback is that the server that allocates the non-DEVS model extended by Matlab functions needs to reference the Matlab MCR. A complete demonstration of DEVS/SOA running a DEVSJAVA, xDEVS and DEVS.NET hybrid model can be found at [15].

4. CASE STUDY

In this section we describe a model of a barrel filler that embraces interoperation among distributed DEVS and non-DEVS models implementations. Filling a barrel is a typical engineering process that involves both continuous and discrete event archetypes. This experiment is motivated on a simple example that can be found at Zeigler's book [12], but with a different unrelated purpose. Figure 8 shows the system structure, it has one continuous input port *inflow*, and two output ports for observing the contents, one discrete output port, *barrel*, and one continuous port, *cout*. There is also a continuous state variable *contents*. The derivative of the continuous state variable *contents* is established by the input value of *inflow*. Barrels are sent out at the discrete output port *barrel* whenever the variable *contents* reaches the value 10. At that point, *contents* is reinitialized.

In order to build a model of a barrel filler we implemented a coupled model that involves an integrator atomic model with an additional *reset* entrance and a simple model consisting of a threshold level detector.

Even though the integrator atomic model fulfills the mealy discrete time specified system (Mealy DTSS) formalism, it is shown [12] how these systems can be specified as DEVS models. A mealy DTSS is represented as a DEVS in a manner similar to a memoryless function with the difference that the state is updated when receives an input. The system will passivate untill it collects an input. Next, it evolves to the transition state, delivers the output, and in the internal transition function computes the next state and makes $\sigma = \infty$, waiting for the arrival of the new period. In the barrell filler sample, the integrator block comprises the state variable *contents*, the two input port *inflow* and *reset* and the output port *cout*. Computes next state handling the integration function parameterized by the *inflow* value reliant on the *reset* entrance value within the internal transition.

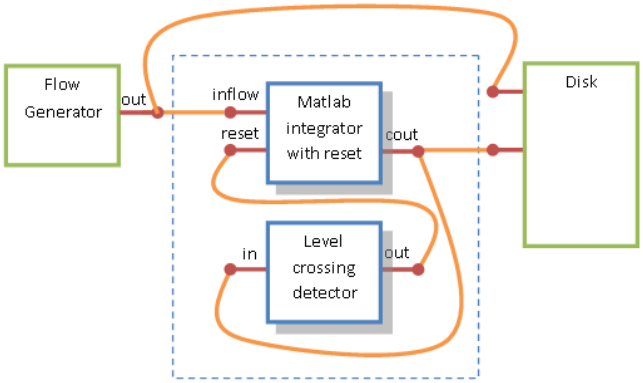

Figure 8. Barrel Filler Model

$$Integrator = \langle X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda \rangle \quad (3)$$

where,

- $X = \{inflow \times reset | inflow \in \Re, reset \in \{true, false\}\}$
- $S = \{contents | contents \in \Re\}$
- $Y = \{cout | cout \in \Re\}$
- $\delta_{ext} : \sigma = h$
- $\delta_{int} : \sigma = \infty$
  - case reset:
    - $false : dcontents/dt = inflow$
    - $true : contents = 0$
- $\delta_{con}(s, \oslash) = \delta_{int}(s)$
- $\lambda : cout = contents$
- $ta(s) : \sigma$

In contrast, the atomic model that embodies a threshold level detector (state event trigger) exemplifies DEVS formalism throghout a a DEVS model. The level detector model only updates the internal event state variable to *true* and triggers the internal transition function ($\sigma = 0$) whenever the input port value crosses the threshold level. Right before the internal transition passivates, the system delivers the state event value throgh the output port.

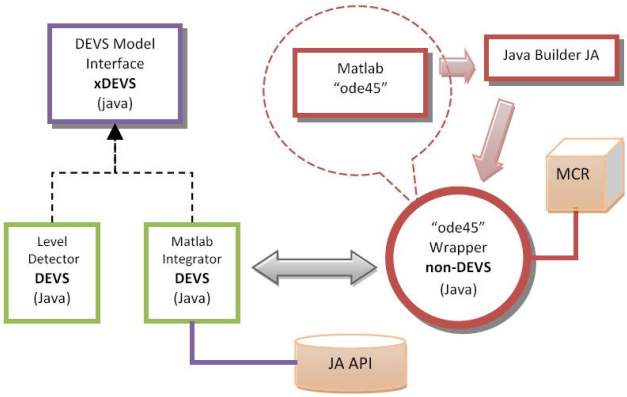

Figure 9. Global Implementation Overview

$$Level \quad = \quad \langle X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda \rangle \qquad (4)$$

where,

- $X = \{in | in \in \Re\}$
- $S = \{state\ event \in \{true, false\}\}$
- $Y = \{out \in \{true, false\}\}$
- δ_{ext} : case reset:
  - $in \geq 10 : state\ event = true,\ \sigma = 0$
- $\delta_{int} : \sigma = \infty$
- $\delta_{con}(s, \oslash) = \delta_{int}(s)$
- $\lambda : out = state\ event$
- $ta(s) : \sigma$

At last, as illustrated on Figure 8, to complete the coupling we link the output port *cout* of the Integrator with the level detector input port and the output port of the level detector component with the input port *reset* of the Integrator module.

Now, the main goal is to segregate the integration duty ahead to the Matlab computing environment. On our first step, with Matlab Java Builder JA support, we generated a Java wrapper that embraces the Matlab integration function "ode45" as shown in Figure 9. With the xDevs model interface as the skeleton, along with previous integrator and level detector DEVS models description, without leaving away Matlab immerse functionality, we are ready to simulate the coupled interoperable model example.

Following Figure 4 steps and Java based DEVS/SOA simulator servers. First we pick out the servers to distribute the complete barrel filler model simulation. Then the combined model is uploaded and compiled in the selected servers. The

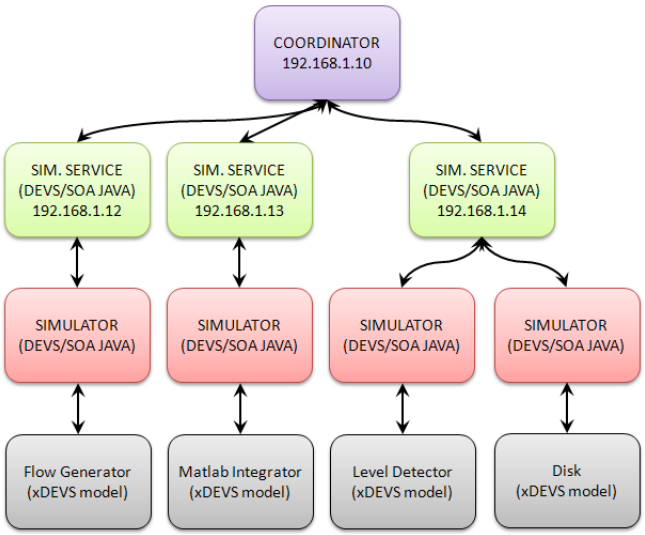

Figure 10. Distributed DEVS/SOA of a Barrel Filler

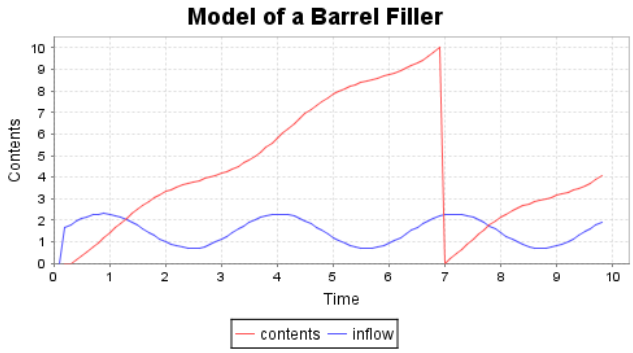

Figure 11. Execution performance of the Barrel Filler Model

main server creates the coordinator and in turn the coordinator creates the corresponding simulators. Finally, as presented on Figure 10, the simulation is executed as a distributed coupled DEVS model with Matlab engaged operability.

To illustrate the execution performance of the above example, we have drawn a graph (see Figure 11). The *X* axis posts the time elapsed, while the *Y* axis presents the measure unit for the barrel filler inflow and the internal variable *contents* status value. The blue series pictures the inflow value at time "t", while the red series illustrates the value of the internal variable *contents* at time "t + h". Whenever the *contents* reaches the tolerance limit level *10*, the level detector component triggers an state event on the way to rearrange the Integrator internal initial value to *0*. As can be seen on Figure 11 we attach a source generator to the model input which propagates the oscillation of a *sine* function and impacts on the deviation of the internal variable *contents* value.

5. CONCLUSION AND FUTURE WORK

The approach proposed in this paper, expects to be an initial implementation for the application of recently developed interoperability standards. We focused our efforts in the need for modeling and simulate atomic and coupled models that share the same semantics given the DEVS mathematical specification, but differ in the computing environment basis. We showed a simulation involving a DEVS model implementation developed with the native programming language of a DEVS simulator environment framework (xDEVS) along with a non-DEVS model that benefits from functions encoded on Matlab programming language. Furthermore, we described an approach supporting interoperability among DEVS models and Matlab functionality in consort with a web-based distributed simulation framework handling The *Service oriented Architecture (SOA)*.

In spite of the work presented on this paper and others [8], interoperability over cross-platform DEVS models implementations relies somewhere within a long distance run early stages, towards a full DEVS capacity inter-operation. In regard to the approach discussed in this research work, the most tempting future objective might be the development of a distributed or non-centric simulation framework for independent Matlab functional blocks as DEVS models combined with DEVS/SOA simulation engine. Matlab operational DEVS simulation engine throughout the *World Wild Web* wraps another level uppon the interoperability standard. However, in this work we provided new technologies to support interoperability between DEVS and non-DEVS based models, developing a platform-independent simulation layer by means of our DEVS/SOA framework. Since the xDEVS interface was used as a "standard" at the modeling layer in this article, our future work includes the extension of our DEVS/SOA architecture to this layer.

REFERENCES

[1] Wikipedia, Available at: http://en.wikipedia.org/wiki/.

[2] S. Mittal, B. P. Zeigler, and J. L. Risco-Martín, "Implementation of formal standard for interoperability in M&S/Systems of Systems integration with DEVS/SOA," *International Command and Control C2 Journal, in review*.

[3] Y. K. Cho, X. Hu, and B. P. Zeigler, "The RT-DEVS/CORBA environment for simulation-based design of distributed real-time systems," *Simulation*, vol. 79, pp. 197–210, 2003.

[4] B. P. Zeigler, S. B. Hall, and H. S. Sarjoughian, "Exploiting HLA and DEVS to promote interoperability and reuse in lockheed's corporate environment," *SIMULATION*, vol. 73, no. 5, pp. 288–295, 1999.

[5] S. Mittal, B. P. Zeigler, J. L. Risco-Martín, F. Sahin, and M. Jamshidi., *Systems of Systems Engineering*. John Wiley & Sons, 2008, ch. Modeling and Simulation for Systems of Systems Engineering, pp. 101–149.

[6] A. Tolk and J. A. Muguira, "The levels of conceptual interoperability model (LCIM)," *Proceedings Fall Simulation Interoperabiliry Workshop*, 2003.

[7] M. J. DiMario, "Systems of systems interoperability types and characteristics in joint command and control," *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, April 2006.

[8] T. Wuztler and H. S. Sarjoughian, "Simulation interoperability across parallel DEVS models expressed in multiple programming languages," *Proceedings of the DEVS Integrative M&S Symposium*, 2006.

[9] "Arizona Center of Integrative M&S (ACIMS)," http://www.acims.arizona.edu, 2008.

[10] S. Mittal, J. L. Risco-Martín, and B. P. Zeigler, "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, Spring Simulation Multi-Conference*, 2007.

[11] S. Mittal, B. P. Zeigler, and M. H. Hwang, "XML-based Finite Deterministic DEVS (XFD-DEVS)," http://www.saurabh-mittal.com/fddevs.

[12] B. P. Zeigler, T. Kim, and H. Praehofer, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000.

[13] G. Wainer, "CD++: a toolkit to develop devs models," *Softw. Pract. Exper.*, vol. 32, no. 13, pp. 1261–1306, 2002.

[14] J. L. Risco-Martín and J. M. Cruz, "xDEVS: DEVS java API," http://www.dacya.ucm.es/jlrisco.

[15] J. L. Risco-Martín, "Demo of interoperability in DEVS/SOA," http://www.dacya.ucm.es/jlrisco.

[16] S. Mittal, J. L. Risco-Martín, and B. P. Zeigler, "DEVS-based web services for net-centric T&E," in *Summer Computer Simulation Conference, SCSC 2006*, 2006.

[17] S. Mittal, J. L. Risco-Martin, and B. P. Zeigler, "DEVS/SOA: A cross-platform framework for net-centric modeling and simulation using DEVS," *Submitted to SIMULATION: Transactions of SCS, in review*, 2007.

[18] "The Mathworks," http://www.mathworks.com/.

[19] B. P. Zeigler and P. Hammonds, *Modeling and simulation-based data engineering: introducing pragmatics into ontologies for net-centric information exchange*. Elsevier Academic Press, 2007.