2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications

DEVS/SOA: Towards DEVS interoperability in distributed M&S

Alejandro Moreno*, José L. Risco-Martín†, Eva Besada‡, Saurabh Mittal§ and Joaquín Aranda*
*Departamento de Informática y Automática
E.T.S.I. Informática (UNED)
Email: amoreno@bec.uned.es, jaranda@dia.uned.es
†Departamento de Arquitectura de Computadores y Automática
Facultad de Informática (UCM)
Email: jlrisco@dacya.ucm.es
‡Department of Computer Science
University of New Mexico
evabes@cs.unm.edu
§Dunip Technologies
P.O. Box 26218, Tempe, AZ 85285, USA
Email: saurabh.mittal@duniptechnologies.com

***Abstract*—DEVS *Modeling and Simulation* (M&S) has multiple implementations with several computer languages such as Java, C# or C++. Therefore emerges the need of a distributed platform to provide interoperability mechanics for simulation and encourage reusability of legacy simulations and integration of diversified DEVS models. In this paper, we apply a recently proposed interoperability standard for DEVS M&S throughout a renewed version of DEVS/SOA. The main goal of this web oriented framework is to connect heterogeneous DEVS simulation elements in a transparent, open, and scalable way. We define a DEVS/SOA implementation that embodies *Service Oriented Architecture* (SOA) within WSDL standard to describe the simulator and coordinator interfaces and SOAP standard to support communication operations between them. This arrangement allows frontend user applications to lead simulations without local access to modeling components. Furthermore, we illustrate a real military based example of DEVS simulation interoperability among Java and .NET based DEVS models. Experiments in two different examples show that when the simulation is distributed using DEVS/SOA, we obtain a speed-up of 22% in average.**

***Keywords*-Distributed Simulation, Interoperability, DEVS, Service Oriented Architecture.**

I. Introduction and related work

The simulation field in computing, attempts to simulate an abstract model of a particular system. Computer simulation is a useful part of mathematical modeling of many natural systems in physics, chemistry and biology, human systems in economics, psychology, and social science and in the process of engineering new technology, to gain insight into the operation of those systems, or to observe their behavior [1]. Computer simulations vary from computer programs that run a few minutes, to network-based groups of computers running for hours, to ongoing simulations that run for days. The human capacity required for modeling complex systems has far exceeded anything feasible for independant research labs and the scale of simulated events for computer simulations lack of computational resources. For this reason, ambitious and challenging tasks in the simulation society claim for global support. In this context, there are various arguments to back up the need of a "universal" tool or framework that embraces simulation of heterogeneous models: reusability of legacy simulation; access to remote modeling components; limited human and computational resources; integration of models with regards to software platforms; and the avoidance of superfluous or redundant work. In order to fulfil these requirements, first of all, we propose the Internet as the global data communication infrastructure to provide connectivity between computers due to it's ubiquity, as well as the *World Wide Web* (WWW) to supply simulation services via the Internet. In addition to this, we employ the *Service-Oriented Architecture* (SOA) to provide methods for systems development and integration where systems package functionality as interoperable services. Moreover, these services satisfy uniform technical criteria established by standards to tackle any interoperability difficulties. Consequently, we apply the *Web Service Description Language* (WSDL) standard to describe the services interface and the *Simple Object Access Protocol* (SOAP) standard to support communication between them. The application of these standards over the proposed architecture enables the platform-independent deployment of simulation services through the Internet.

The DEVS M&S formalism [2] provides several advantages to analyze and design complex systems: completeness, verifiability, extensibility, and maintainability. DEVS can reproduce the other major *Discrete Time System Specification (DTSS)* and approximate continuous modeling paradigms (*Differential Equation System Specification (DESS)*), i.e., DEVS is able to describe discrete event, discrete and continuous systems. Furthermore, DEVS separates conceptually models from the simulator what makes possible to simu-

1550-6525/09 $26.00 © 2009 IEEE
DOI 10.1109/DS-RT.2009.18

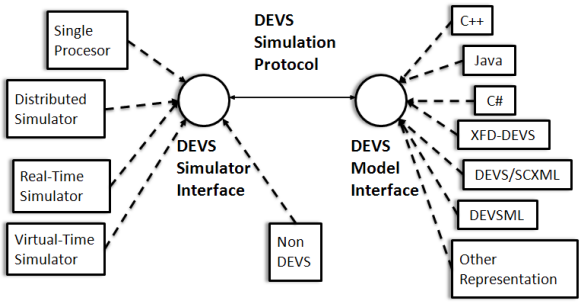

Figure 1. Conceptual Architecture of Standard

late the same model using different simulators working in centralized, parallel or distributed execution modes. Finally, there are several DEVS M&S implementations based on various computer languages such as Java, C++ or C#.

Our initial framework [3], DEVS/SOA, is a distributed M&S tool that provides distributed DEVS simulation services using the aforementioned standards and architecture. The present framework in this paper has been developed within two software platforms: Java and .NET framework that let us work with any type of language taking advantage of the *Common Intermediate Language (CIL)*. Moreover, DEVS/SOA is capable of running a simulation that embraces interoperation of diversified DEVS models encoded in different programming environments such as xDEVS[4], DEVSJAVA[5] or CD++[6] models, and consequently provide multiple DEVS modeling environments to specify different components of the whole system using existing models or creating new ones.

Recently, within a working group of the Simulation Interoperability Standards Organization, a standard [7] 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 applications, 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 [5]. 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.

There are several current DEVS implementations providing distribution and interoperability employed at different levels. JAMES constitutes a framework which is aimed at supporting experiments with agents under temporal and resource constraints. Its core libraries provide the means for the description of variable structure models and their distributed, parallel execution [8]. Even though this study provides insteresting formal aspects of parallelization, JAMES does not support interoperability. Regarding interoperability, the DEVS/CORBA [9] 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 [10] simulation environment supports high level model building using DEVS methodology as well as supplying heterogeneous simulation models interoperability based on HLA concepts. While DEVS/CORBA and DEVS/HLA are parallelized frameworks based on middleware technologies, the framework proposed in this paper achieves the distributed simulation by the use of standardized web services and implicitly assume the interfacing and coordination DEVS Simulation Protocol Standard proposed in [11]. In addition to this, recently a CD++ [6] simulation framework that provides Web Services according to the REST principles RESTful-CD++ [12] has been presented. However SOA based Web Services overtakes REST based Web Services due to several reasons: SOA is transport agnostic, is designed to handle distributed computing environments, is the prevailing standard for web services, has built-in error handling and provides extensibility; REST assumes a point-to-point communication model that is not usable for distributed computing environment where message may go through one or more intermediaries, lack of standards support, and is tied to the *Hypertext Transfer Protocol (HTTP)* transport model. Furthermore, there are other distributed simulation approaches that employ SOA as the communication pattern design framework that enhance their performance [13] or discuss the motivation for developing such a framework [14], but they do not enclose DEVS M&S formalism. The advantages of our approach are the following. First, the original models, using native DEVS libraries, can be distributed without middleware. Second, the models of several DEVS platforms such as xDEVs, DEVSJAVA or CD++ can be combined to define the whole system providing interoperability. Third, different simulation services implemented in different programming languages for the different modeling platforms can be distributed through the Internet and connected using the WSDL and SOAP standards, allowing a friendly-user deployment of the simulation of the whole system through the Internet. Finally, this approach, which achieves the design pattern of DEVS separation between modeling and simulation, could incorporate other modeling paradigms and simulation services, extending the interoperability of DEVS to other techniques [15].

This work is encouraged to satisfy and promote the formerly stated interoperability standard for DEVS M&S. Unfortunately, the standard lacks of a formal detailed particularization of DEVS simulator and model interface. Therefore we make use of our DEVS/SOA common interfaces framework as a point of union among distinct DEVS mod-

eling and simulation platforms. In addition, we demonstrate the application of DEVS/SOA net-centric modeling and simulation environment with a real military based case study.

II. Background

A. 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 [2], chapter 4. 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 $\delta_{con}(s, \emptyset) = \delta_{int}(s)$ [2]
- $\lambda : S \rightarrow Y$ 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”, 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).

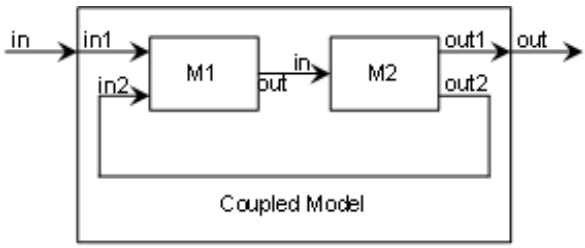

Figure 2. Coupled DEVS model

There are varied libraries for expressing DEVS models across the globe, such as DEVSJAVA [5], DEVS/C++ [5], CD++ [6], xDEVS [4], 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 [2].

III. Approach

A. Introduction

Web applications are typically described as cross-platform because, ideally, they are accessible from any of various web clients on different operating systems. However, such applications diversify widely in complexity and functionality. This wide variability inhibits the goal of multi-platform interoperation capacity. In order to address DEVS Web oriented applications interoperability weakness, we applied the newly proposed interoperability standard for DEVS Modeling and Simulation [7]. Unfortunately, the standard lacks of a formal detailed particularization of DEVS simulator and model interfaces. To tackle this inconvenience, the approach described on this paper implements our DEVS/SOA common interfaces framework as a point of union among distinct DEVS modeling and simulation platforms. Besides, the standard omits a third degree of interoperability, the interaction among a third party component, DEVS Coordinators along with DEVS Simulator engines. As shown in Figure 3 we adopt the same structure and behavior as for DEVS Modeling and Simulation standard to include DEVS simulators coordination as another level of interoperability.

Figure 3 illustrates standard distributed simulation among different DEVS simulation and coordination engines supported by DEVS/SOA. A multi-platform simulation that

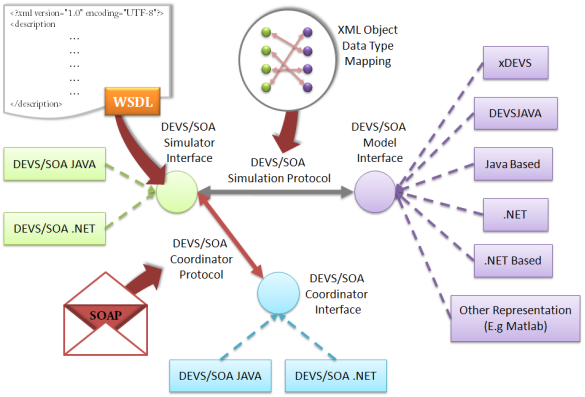

Figure 3. Application of the Standard

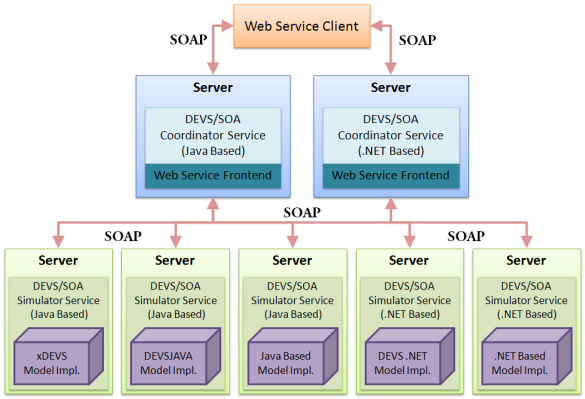

Figure 4. DEVS/SOA Distributed Architecture

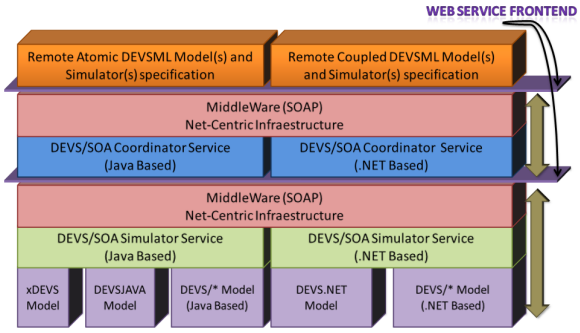

Figure 5. DEVS/SOA Architecture

comprehends a Java sustained simulation environment together with a .NET based simulation framework. Since both simulation platforms are based on web services, 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. Furthermore, Figure 3 depicts the use of WSDL to provide DEVS Simulator Service operation description as DEVS Simulator Interface. The Simulator Interface specification is independent of the specific technological platform used to implement it. As a consecuence of Web Services implementation, Figure 3 remarks the usage of SOAP standard technology as the communication protocol among simulators and coordinators. Moreover, as seen on Figure 3, we incorporated *Extensible Markup Language* (XML) object data type mapping as message exchange format for DEVS/-SOA Simulation protocol among simulators. To conclude the analysis of the overall approach enclosed in the SISO DEVS M&S standard [7], we center our focus towards DEVS modeling interface pictured in Figure 3. Although rigorously speaking, there is not a formal standard DEVS model interface, in this work, we propose our DEVS/SOA Modeling interface as a railway junction between different simulation modeling frameworks. The DEVS/SOA Model interface describes DEVS model operations encoded both in Java for xDEVS[4], DEVSJAVA[5], and Java based DEVS models, and in .NET for DEVS.NET and .NET based models.

B. Architecture

In this work, we developed an application of the standard within the Service Oriented Architecture (SOA) providing DEVS modeling and simulation services over the World Wide Web. The major advantage of this architecture for interoperability upon systems, is the unawareness of simulators and coordinators platform among themselves and other Web Service Clients. Furthermore, the DEVS/SOA simulator provides a model interface and XML object mapping for message exchange that enables the integration of models expressed in distinct DEVS M&S libraries (DEVSJAVA, xDEVS, and Java based models in parallel with DEVS.NET and .NET based models).

In a distributed simulation accomplished within DEVS/-SOA simulation framework a DEVS coupled model is split up into models simulators that run on multiple computers communicating over a network. A DEVS/SOA coordinator deals with simulators hosted anywhere over the web. The main goal of this achievement is to connect DEVS simulation elements in a transparent, open, and scalable way. Ideally this arrangement is drastically more powerful than many combinations of stand-alone computer simulations primarly because frontend user applications may lead distributed simulations without local access to modeling components. DEVS/SOA distributed computing nature is depicted at Figure 4.

The proposed DEVS/SOA architecture embraces essentially three levels of interoperability upon DEVS M&S elements. The outer level comprehends the interaction among the application layer and DEVS coordination service within SOAP communication protocol and the Coordinator WSDL description file. Beyond this, we encounter a similar interoperation level, intercommunication between DEVS coordination service and DEVS simulation service. Again, we

apply SOAP protocol specification for exchanging structured information between DEVS coordinators and simulators. The operations offered by the Simulator service are written in the Web Services Description Language. Even more deeply across DEVS/SOA architecture we come upon a third level of interoperability, the junction of a DEVS simulator and the associate DEVS model. This union is bounded by the DEVS/SOA modeling interface and messages between DEVS models are expressed in XML. The basics of DEVS/SOA architecture is captured in Figure 5.

The top of the layered architecture is the application layer that contains remote models specification in DEVSML[16]. These models encoded in XML language provide a detailed remote characterization of a DEVS compound model. Figure 6 describes throughout DEVSML a simple *Experimental Frame (Generator-Transducer) - Processor* (EFP) DEVS model implementation. The root coupled element determines the identification and host of the coupled model. This host must provide a DEVS coordination service accordant with our DEVS/SOA WSDL operations. The root coupled element children may identify a DEVS atomic model, a connection among DEVS models, input and outptu ports, or in a recurrent manner, another DEVS coupled model. Each atomic model item must denote a series of characteristics: the name of the atomic model; the platform where the model is able to perform; the class that implements the model and must be instantiate; the host which determines the localization of the server that provides the simulation service accordant with our DEVS/SOA WSDL operations and enjoys access to the model implementation class repository; and finally as internal elements, the model provides input and output ports specification along with their class type messages. Each connection element designates a link between a model output port and a model input port. The message types among connections need to be fully compatible (involving different platforms), otherwise, the overall simulation might fail.

However, as illustrated at Figure 5 the client application requires a Web Service frontend to provide access to the Coordination Service layer throughout the standard SOAP protocol, in regard of software platforms. Assembled with a Web Service framework, the client application acquires the capacity to search over the World Wide Web for the coordinator service hosted at a remote location stated at the DEVSML root model. If the connection is established, the web client application may activate the remote coordinator by suppling a DEVSML based model description and subsequently command the simulation accordant to specific client parameters. As soon as the simulation culminates, the client application receives in exchange a summarize of the overall simulation performance. The aforementioned operations are understood inside (Listing 1) the coordination interface for the DEVS/SOA Coordinator in concordance with the DEVS/SOA Coordinator Service WSDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<coupled name="efp" host="http://192.168.1.2:8080/axis2/services/Coordinator">
 <coupled name="ef" platform="xdevs" class="Ef" host="http://...">
  <atomic name="gen" platform="xdevs" class="Generator" host="http://...">
   <inport name="stop" class="Job"/>
   <outport name="out" class="Job"/>
  </atomic>
  <atomic name="tran" platform="xdevs" class="Transducer" host="http://...">
   <inport name="solved" class="Job"/>
   <inport name="arrived" class="Job"/>
   <outport name="out" class="Job"/>
  </atomic>
  <inport name="in" class="Job"/>
  <outport name="out" class="Job"/>
  <connection atomicFrom="ef" portFrom="in" atomicTo="trans" portTo="solved"/>
  <connection atomicFrom="gen" portFrom="out" atomicTo="ef" portTo="out"/>
  <connection atomicFrom="gen" portFrom="out" atomicTo="trans" portTo="arrived"/>
  <connection atomicFrom="trans" portFrom="out" atomicTo="gen" portTo="stop"/>
 </coupled>
 <atomic name="processor" platform="devs.net" class="Processor" host="http://...">
  <inport name="in" class="Job"/>
  <outport name="out" class="Job"/>
 </atomic>
 <connection atomicFrom="ef" portFrom="out" atomicTo="processor" portTo="in"/>
 <connection atomicFrom="processor" portFrom="out" atomicTo="ef" portTo="in"/>
</coupled>
```

Figure 6. ef-p DEVS/SOA model

Listing 1. DEVS/SOA Coordindator Interface

```
public interface Coordinator{
   public void setModel(String XmlModel);
   public String[] simulate(int numIterations);
}
```

Likewise, as pictured in Figure 5, the Coordinator service layer assembled with a Web Service Frontend gains the capacity to invoke DEVS simulation operations against the Simulator Service layer. The communication is driven by the standard SOAP protocol, once again independently of software frameworks. Moreover, assuming that a coordinator is already provided with the root DEVS model, the initial coordination task comprehends parsing the DEVSML Document searching for DEVS models and connections among themselves. Each XML encoded model of the DEVSML document supplies the coordinator with the associate remote simulator location in order to enable a communication among them. What's more, the connections between models capacitates the coordinator to be aware of the whole distributed circuit so as to carry out a proper execution. Whenever the web linkage among layers is set up, the coordinator proceeds to activate all simulators by feeding them with the corresponding DEVSML model description. Afterwards, the coordinator continues with the simulation ignition within the DEVS/SOA simulation protocol. Technically, these procedures are expressed in the simulation interface of DEVS/SOA illustrated at Listing 2 that stands for the Web Service Description Language file of the DEVS/SOA Simulator Service. These operations wrap DEVS simulation protocol with the purpose of achieving a qualified DEVS simulation.

Listing 2. DEVS/SOA Simulator Interface

```
public interface Simulator{
   public void setModel(String XmlModel);
   void initialize(double t);
   double getTN();
   void deltfcn(double t);
   void lambda(double t);
   String[] getOutput(String portName);
   void receive(String portTo, String[] xmlMessage);
}
```

Furthermore, the DEVS/SOA simulator integrates a common DEVS model interface pictured at Listing 2 to support multi-framework interoperability. This modeling interface plus a customized adapter if needed allows the aggregation of Java based models (xDEVS and DEVSJAVA) in parallel with .NET based models (DEVS.NET). A noteworthy achievement of our DEVS modeling common interface is the presence of an embedded "translator", a module in charge of mapping incoming messages to XML and outcoming messages the other way around.

Listing 3. DEVS/SOA Atomic Interface

```
public interface AtomicInterface{
    void initialize();
    //DEVS time advanced function
    double ta();
    //DEVS internal transition function
    void deltint();
    //DEVS external transition function
    void deltext(double e);
    //DEVS confluence funtion
    void deltcon(double e);
    //DEVS output function
    void lambda();
    //maps the arriving XML formated Messages to object
    //Messages and loads it in port 'portTo'
___void receive(String portTo, String[] xmlMessage);
___//maps the outcoming Messages at port 'portName' to XML
    String[] getOutput(String portName);
}
```

In distinction from DEVS mathematical formalism for discrete event systems, our DEVS model interface exemplified at Listing 2 gathers the basic function definitions of DEVS system specification except for the external transition and confluence function. Our definition of the external transition and confluence function differs from the classic DEVS resolution in the absence of a parameter as a representative of an input message. That is just because the external message is already loaded at some inport of the concrete DEVS model (see function receive at Listing 2) and there is no need to add reiterative parameters to any function skeleton.

In other respects, the essential duty of the aforementioned DEVS multi-framework modeling interface translator is binding XML data types along with message objects between the DEVS/SOA simulator and the DEVS models in regard of their platform. This procedure converts a DEVS message (DEVS model input or output) built from class objects into an XML based document (marshall), and vice versa (unmarshall). Meanwhile, Figure 7 illustrates how the XML message binding process affects the global simulation process. Due to this XML serialization, DEVS models inputs and outputs are sent within an standardized context throughout the SOAP protocol. Nevertheless, the major drawback of our architecture is the quest for a wrapper library that allows storing and retrieving data in memory in any XML format, without the need to implement a specific set of XML loading and saving routines for the program's class structure. Since the purpose of our architecture is the interoperation among different platforms, the XML data types mapping of each framework must correlate, e.g. a *String* written whether in Java or .NET needs to be mapped to the same XML Schema

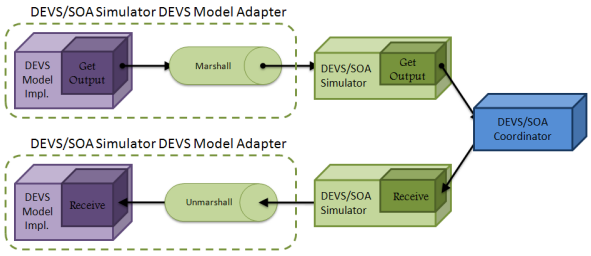

Figure 7. DEVS/SOA XML message Binding in the overall Simulation

Type *xsd:string*. In this work, we employed *Java Architecture for XML Binding* (JAXB) that allows Java developers to map Java classes to XML representations together with the *XmlSerializer*, i.e. a .NET Framework class that serializes and deserializes objects into and from XML documents and enables you to control how objects are encoded into XML. Fortunately, Both *Application Programming Interfaces* (API) that provide XML data type binding are highly compatible for standard data types.

Whenever a distributed simulation is carried within DEVS/SOA, the coordinator is in charge of launching each simulator sending them the XML model specified by DEVSML. Afterwards, the coordinator and simulators begin the typical message passing process defined in DEVS. Figure 8 illustrates a use case of DEVS/SOA Simulation platform from a global perspective. Firstly, the Coordinator Services receives the whole XML model specification. Next, the Coordinator extracts each DEVSML child model description, transfers it to the associated simulator service, and stores the connections among models. As soon as the Simulator receives the model description, it creates an instance of the identified DEVS model stored at an accessible repository in order to begin the simulation. As highlighted in Figure 8, the simulation ignition holds back until the Coordinator provides the corresponding DEVSML model description for each Simulator Service. Afterwards, the simulation functionality basics are carried.

DEVS/SOA is a simulation framework founded on SOA that offers DEVS-based simulations as Web Services, which are based on standard technologies (SOAP and WSDL). As a result of the employment of these standards, we obtain formal documents that establish uniform Web Service description of DEVS simulation and coordination service. In fact, the simulation procedures translated to standard description language for Web Services provides the essential guidance for the final implementation of DEVS service oriented simulators and coordinators internal behavior. Technically, as shown in Figure 9, we applied two different platform dependant tools to generate the code skeletons of our implementations: Axis2/Java core engine for Java based Web Services and the *Web Service Enhancement* (WSE) tool for .NET based Web Services. Although these procedures might

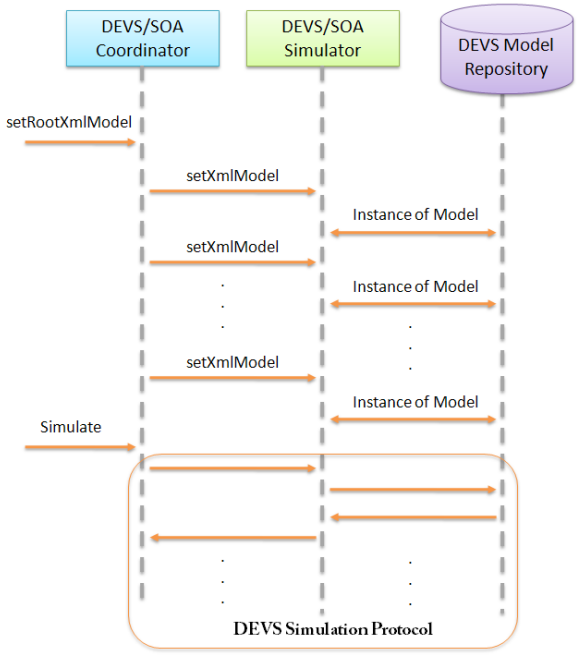

Figure 8. DEVS/SOA Use Case

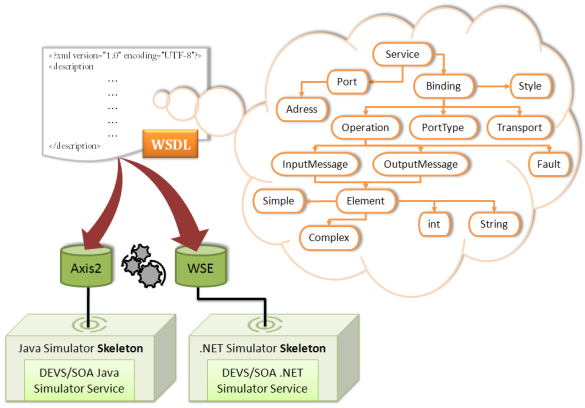

Figure 9. DEVS/SOA Simulator WSDL provides DEVS Simulator Interface

seem to move against interoperability principles, both tools depart from the same service description language document and therefore the outcome performance is homogeneous.

C. Integration of non-DEVS models

In other respects, because of the abstract or unexplicit nature of DEVS/SOA model interface, arises the possibility of integration of non-DEVS models by the means of an adapter, i.e. the Matlab simulation environment as a non-DEVS model[15] can be applied over DEVS/SOA modeling interface. Likewise, for example the interoperation with an unrelated Web Service may be as a non-DEVS coupled model[17].

IV. Example

DEVS/SOA has been designed not only to allow us interoperability, but also to improve the performance of complex simulation models when they are distributed. Next, we show such characteristics in the simulation of *Unmanned Aerial Vehicles (UAVs)* in hostile environments.

Unmanned Aerial Vehicles are aircrafts without onboard pilots that can be remotely controlled or fly autonomously based on pre-programmed flight plans [18]. They can be used in a wide variety of fields, both civil and military, such as surveillance, reconnaissance, geophysical survey, environmental and meteorological monitoring, aerial photography, and search-and-rescue tasks. In military missions they work in dangerous environments, where it is vital to fly along routes which keep the UAVs away from any type of threat and prohibited zones. The best routes are those which minimize the risk of destruction of the UAV and optimize some planning criteria (such as flying time and path length) while fulfilling all the physical constraints of the UAVs and its environment, plus the restrictions imposed by the selected mission.

In [19], authors present a novel path planner for this kind of systems. Based on evolutionary computation, it is able to find a list of 3D points, also called *WayPoints (WPs)*, which are used to compute (offline) a spline curve [20] that constitutes the solution of the problem: the 3D path of the UAV. Such path is validated against a simulator. However, the planner must be able to also work online in order to propose a new path during the mission of the UAV when a pop-up (unknown threat) appears. In the current research work, we study the performance of such simulator when it is implemented using DEVS (in a classic manner) and DEVS/SOA in an heterogeneous model. It means that every component of the DEVS/SOA system is implemented using different DEVS simulation platforms. In addition, we integrate an online path-planner in the UAV model (i.e., the UAV model includes an embedded radar). Thus, when a pop-up is detected, the UAV must compute a new path to avoid the risk. It takes several seconds before the UAV can modify its original trajectory, adding extra computational effort at simulation time. However, such extra effort makes the system more realistic and it can be reduced when the simulation is distributed. To test interoperability between several simulation engines we developed the same original models in [19] using DEVSJAVA (Java) and DEVS.NET (.NET/C#). After that, we distributed the simulation over several computers to check the performance of parallelization.

Next, we describe our DEVS model and how it is distributed using DEVS/SOA. Finally we present the results.

A. DEVS/SOA model for UAVs simulation

Figure 10 depicts the DEVS simulation model. It is formed by two coupled models: the UAV and the *Air Defense*

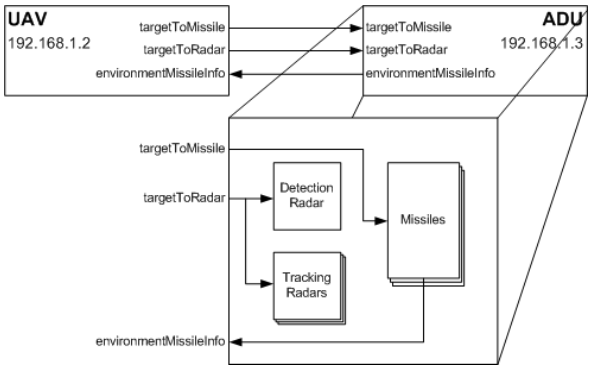

Figure 10. DEVS root coupled model for UAVs simulation. Boxes are models, arrows are couplings and labels are input and output port names. Every atomic or coupled model can be placed at different computers.

Unit (ADU). We can include as many UAVs and ADUs as needed, and all of them follows the structure shown in Figure 10.

On the one hand, at every integration step a UAV must send to every ADU in the example some parameters to check if a radar has detected the corresponding target or a missile has hit upon it. Such parameters includes position, velocity, and Euler angles. On the other hand the ADU must send to every UAV some information regarding the state of the missiles (if they have been exploded or not, for example).

When integrating the DEVS model into our DEVS/SOA framework, the original root coupled model must be described in terms of DEVSML to perform the distributed simulation (see Section III-B). At this level, every atomic model can be developed using any of the simulation platforms supported by DEVS/SOA (DEVSJAVA, xDEVS, MicroSim/Java or DEVS.NET are allowed). In our case we are evaluating the distribution of several scenarios using xDEVS, DEVSJAVA and DEVS.NET, as we show in our case study. To prove the interoperability of DEVS/SOA we are performing simulations of 3 UAVs (using DEVSJAVA, xDEVS and DEVS.NET respectively) and several ADUs (using xDEVS) depending on the mission under study.

We have set up a suite of experiments to analyze performance aspects of DEVS/SOA for this particular application. Three different scenarios are considered. In the *Baseline* scenario all components (3 UAVs and several ADUs, depending on the experiment) are simulated using xDEVS in a single operating system. In the *Local* scenario, all components are simulated using DEVS/SOA in a single operating system, in such a way that each UAV is simulated using xDEVS, DEVSJAVA and DEVS.NET, respectively. In the *Heterogeneous* configuration, there exist 3 computers, one using a Linux OS and the other two under Windows O.S. Every computer simulates one single UAV. On the other hand, ADUs are spread among computers involved. In the set of experiments shown, all the ADUs are configured as pop-ups.

Next we show the configuration of the two evaluated examples. After that, we show the performance obtained when the system is distributed using DEVS/SOA. We see that distributed simulations outperform baseline simulations. In addition, heterogeneous simulations reach the same results in terms of mission success.

B. Experiments

We have configured two different examples to test our distributed simulations. Fig. 11 depicts these examples (called A and B).

Every big blue dashed circles mark the maximum distance of detection for each ADU, while the small red solid ones enclose the zones where the *Probability of Kill (Pk)* is greater than 0.

Every trajectory is labeled by its corresponding UAV (UAV1, UAV2, and UAV3 in our case). The dashed trajectory shows the original one computed by the offline path-planner, whereas the continuous one shows the trajectory computed in simulation time when the UAV's embedded radar detects a pop-up. Thus, each of the two examples in Fig. 11 shows one of the simulations when all the UAVs reach the Goal point.

In our experiments all the ADUs are configured as pop-ups. In addition we have checked in our simulations that no UAV survives when the embedded radar is switched off. When activated, the simulator will compute an alternative trajectory, as soon as the UAV's embedded radar detects a pop-up. In addition we have reduced the maximum range of the embedded radar for Example B.

In Example A (Fig. 11), the embedded radar has been set to its maximum range (50 Km). When one UAV detects a pop-up, there exist a little corridor to escape from it. The same happens in Example B (right side of Fig. 11), where two *No Flight Zones (NFZ)* have been included and represented as rectangular shaded zones. In Example B the range of the embedded radar is not big enough to escape from the pop-ups, so the percentage of success is considerably reduced. Example B also includes a more sofisticated trajectory, in presence of NFZs and where there exist two WPs near of the pop-ups.

C. Results

Figure 12 depicts speed-ups with respect to the *Baseline* scenario. On the other hand, Figure 12a) depicts speed-ups when there are no pop-ups in both examples, i.e., positions of all the ADUs are known at planning time. On the contrary, Figure 12b) shows speed-ups when pop-ups are activated. It shows that parallelization is specially useful when there exist pop-ups, since every UAV must recompute its original trajectory at simulation time.

First, we analyze results when pop-ups are acting as regular ADUs. It should be note that in the *Local* configuration every UAV can be placed in a separated thread

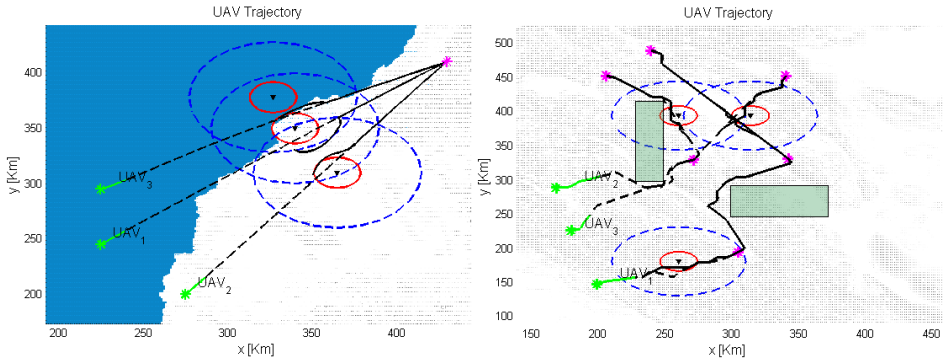

Figure 11. Eagle eye view of the Examples A (left) and B (right).

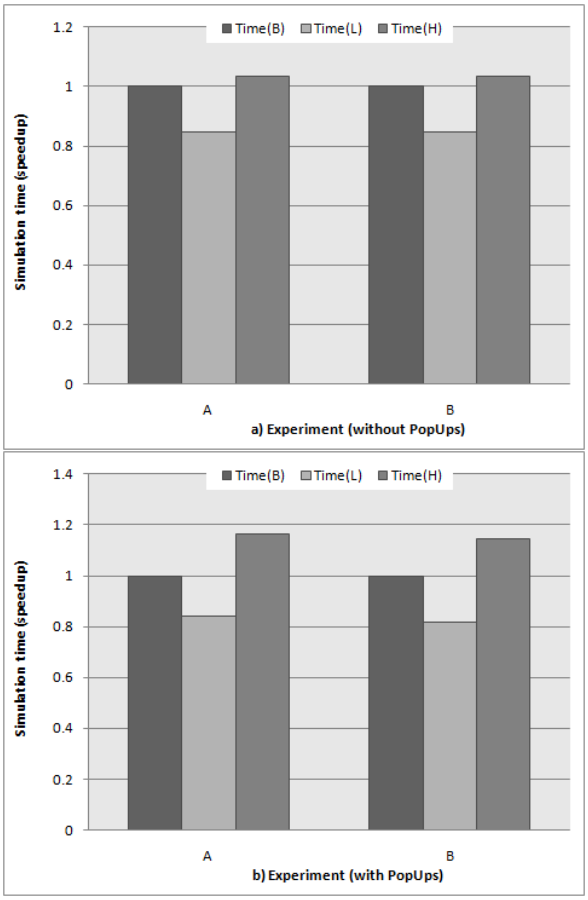

Figure 12. Results.

(due to distributed nature of DEVS/SOA). However, such characteristic has been disabled in order to perform a fair comparison. We can appreciate from Figure 12a) that the worst configuration is given when DEVS/SOA is used in a single machine (*Local* configuration). In particular, as Figure 12a) depicts, it is 16% slower than the *Baseline* configuration in the case of Example A, and 18% in B (17% slower in average). It is logical because, as it has been mentioned before, a lot of communication is needed between UAVs and ADUs. DEVS/SOA is transforming every message from a *Platform Specific Model (PSM)* to XML and vice versa and it requires an extra computational effort. The best performance is reached by the *Heterogeneous* configuration in all the examples. In average, it is 20% faster than the *Local* configuration and 3.30% faster than the *Baseline* scenario: 3% faster in Example A, and the same in B. Once again, even although the system is distributed, the reason of having a small speed-up with respect to the *Baseline* configuration is the intensive communication between all the UAVs and ADUs.

Next, we analyze results when pop-ups are activated. In this case every UAV must compute an alternative route to escape from the pop-ups when they are detected by the UAV's embedded radar. If every UAV detects one pop-up, the distributed system will divide this extra simulation time by three (in the worst case). Thus, summarizing the data depicted in Figure 12b), we can conclude that the Local configuration is still the slowest one (18.3% on average when compared with the *Baseline* configuration). However, in this case we can appreciate a good speed-up when the Heterogeneous configuration is used: the simulator is 22% faster in average: 21% for Example A, and 23% in C. Thus, the DEVS/SOA simulator, when distributed among several machines, is quite better than a classical DEVS simulation.

V. Conclusions and Future Work

In this paper, we described a distributed interoperable DEVS simulator framework DEVS/SOA that deploys simulation and coordination services over the Internet taking advantage of the Web capacity. We applied the service-oriented architecture within the standards: WSDL for the service description and SOAP for ground level protocol communication between the application frontend, simulator and coordinator. This framework is designed and implemented in Java and .NET environments to embody interoperation among DEVS models encoded on the widely-used Java programming language and .NET native languages and due

to .NET's special design, we are able to work with any type of languages by the means of an intermmediate language. Moreover, we implemented an XML data type binding for message passing between simulators that maps objects to XML(marshall) and vice versa(unmarshall). The renewed design of DEVS/SOA is inspired on the recently proposed interoperability standard for DEVS M&S. Finally, we illustrated a real simulation case study in the military field to demonstrate the integration of models framed on distinct DEVS M&S simulation environments. As an extension of the work done, we are looking forward to add on our approach the automatic generation of external Web Services as coupled models [17] and to improve our DEVSML specification language to embrace advanced behavior and data type namespace to define messages between simulators. Besides, as complementary work and a parallel task to our distributed platform, our future work pursuits building a DEVS models and experimental frame repository based on ontologies such as the *Web Ontology Language* (OWL) that is considered one of the fundamental technologies underpinning the Semantic Web [21].

ACKNOWLEDGEMENT

We would like to thank Dr. Jesús M. Cruz and Dr. Bonifacio Andrés-Toro at Universidad Complutense de Madrid, Spain for their support. The contributions of authors José L. Risco-Martín, Alejandro Moreno and Joaquín Aranda in this paper are sustained by the Spanish Ministry of Education and Science projects DPI2006-15661-C02-01 and DPI2006-15661-C02-02, while Dr. Eva Besada-Portas is supported by the Spanish post-doctoral grant EX-2007-0915.

REFERENCES

[1] S. H. Strogatz, "The end of insight," *What is Your Dangerous Idea?*, 2006.

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

[3] S. Mittal, J. L. Risco-Martín, and B. P. Zeigler, "DEVS/SOA: A cross-platform framework for net-centric modeling and simulation in DEVS unified process," *SIMULATION: Transactions of SCS*, vol. 85, no. 7, pp. 419–450, 2009.

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

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

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

[7] B. P. Zeigler, S. Mittal, and X. Hu, "Towards a formal standard for interoperability in M&S/System of Systems integration," *GMU-AFCEA Symposium on Critical Issues in C4I*, 2008.

[8] A. Uhrmacher and K. Gugler, "Distributed, parallel simulation of multiple, deliberative agents," in *PADS*, 2000, pp. 101–108.

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

[10] 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.

[11] K. Al-Zoubi and G. Wainer, "Interfacing and coordination for a devs simulation protocol standard," *IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, 2008.

[12] ——, "Using REST Web-Services Architecture for Distributed Simulation," in *PADS'09: Workshop on Parallel and Distributed Simulation*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 114–121.

[13] A. Park and R. M. Fujimoto, "Aurora: An approach to high throughput parallel simulation," *Principles of Advanced and Distributed Simulation*, 2006.

[14] X. Chen, W. Cai, S. J. Turner, and Y. Wang, "Soar-dsgrid: Service-oriented architecture for distributed simulation on the grid," *Principles of Advanced and Distributed Simulation*, 2006.

[15] J. L. Risco-Martín, A. Moreno, J. M. Cruz, and J. Aranda, "Interoperability between DEVS and non-DEVS models using DEVS/SOA," *Spring Simulation Multiconference*, 2009.

[16] 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.

[17] C. Seo and B. P. Zeigler, "Automating the DEVS modeling and simulation interface to web services," *Spring Simulation Multiconference*, 2009.

[18] B. L. Stevens and F. L. Lewis, *Aircraft Control and Simulation*, 2nd ed. Wiley, 2004. [Online]. Available: http://www.loc.gov/catdir/toc/onix03/91013413.html

[19] J. M. Cruz, E. Besada-Portas, L. Torre-Cubillo, B. Andrés-Toro, and J. A. López-Orozco, "Evolutionary path planner for UAVs in realistic environments," in *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2008, pp. 1477–1484.

[20] G. E. Farin, *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Code*. Orlando, FL, USA: Academic Press, Inc., 1996.

[21] F. V. Harmelen and D. McGuinness. (2004, February) Owl web ontology language overview. W3C. W3C recommendation. [Online]. Available: http://tibor.w3.org/TR/owl-features/