Towards a Formal Standard for Interoperability in M&S/System of Systems Integration

Bernard Zeigler, Saurabh Mittal
Arizona Center for Integrative Modeling and Simulation,
University of Arizona,
Tucson, AS
{zeigler | saurabh} @ece.arizona.edu

Xiaolin Hu
Dept of Computer Science,
Georgia State University,
Atlanta, GA
xhu@cs.gsu.edu

Abstract

Modeling and Simulation (M&S) is finding increasing application in development and testing of command and control systems comprised of information-intensive component systems. In this paper, we apply a System of Systems (SoS) perspective on the integration of M&S with such systems. We employ recently developed interoperability concepts based on linguistic categories along with the Discrete Event System Specification formalism to propose a standard for interoperability. We will show how the developed standard is implemented in DEVS/SOA net-centric modeling and simulation framework.

1. Introduction

Modeling and Simulation (M&S) is finding increasing application in important aspects of command and control systems comprised of information intensive component systems. One aspect of such application is the incorporation of M&S functionality into such systems, an objective of the Extensible Modeling and Simulation Framework (XMSF), a set of Web-based technologies and distributed testbed [1]. Another aspect, the use of M&S to support the development and testing such systems, as instances of System of Systems (SoS). The SoS concept relates to the attempt to integrate disparate systems to achieve a specific goal, typically not co-incident with the goals of the pre-existing component systems. Consequently, the defining concern in SoS engineering is interoperability, or lack thereof, among the constituent system [1, 2]. Achieving such interoperability is among the chief SoS engineering objectives in the development of command and control (C2) capabilities for joint and coalition warfare [3]. Sage [1] analogized the construction of SoS to the federation of socio-political systems and drew a parallel between such processes and the federation that is supported by the High Level Architecture (HLA, an IEEE standard fostered by the DoD to enable interoperation of simulation components [4]). In this light, the present author discussed the role that modeling and simulation (M&S) can play in helping to address the interoperability problems in SoS engineering [5]. The present paper builds upon this work by considering not only the parallel between SoS engineering and distributed simulation, but also how M&S can be more integrally included within SoS engineering approaches. The focus of this paper is to present fundamental concepts to help tackle the integration of M&S and C2 SoS through the use of concepts and standards for interoperability based on the Discrete Event Systems Specification (DEVS) formalism. Our ultimate motivation is to apply M&S concepts and technologies to support collaborative decision making in C2 SoS as well as the testing and evaluation of such systems.

2. Interoperability in Distributed Simulation

As illustrated in Figure 1, 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, as illustrated in Figure 1, are missing from HLA standard as such.

3. Levels of Conceptual Interoperability Model

Although Levels of Information Systems Interoperability [8] models are used successfully to determine the degree of interoperability between information technology systems, they do not

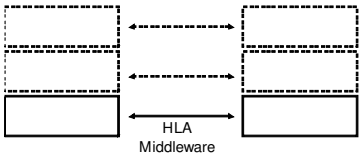

Figure 1. HLA Technical Interoperability

provide a systematic formulation of the underlying properties of information exchange. To remedy this situation, the LCIM outlined in Table 1, was developed to become a bridge between conceptual and technical design for implementation, integration, or federation [9, 10].

The last column lists key conditions that are required to reach an interoperability level from the one below. Of course, the conditions accumulate as the level increases. We note that the conditions given in the LCIM for pragmatic interoperability require that the use of data be mutually understood, where the term "use" is interpreted as the context of its application. A reformulation of LCIM was presented in [11] where more definitive concepts for pragmatic interoperability including the concepts of pragmatic frames and pragmatic equivalence. Moreover, the definition of the semantic level requires the use of a single reference semantic model as a hub for information exchange among participants in collaboration. However such a hub and spokes approach, while desirable, is not always feasible. [12] evaluated a common information exchange model, C2IEDM, as an interoperability-enabling ontology for command and control. The conclusion is that even if there is room for improvements, the model supports almost all basic needs for such a semantic bridge. However, [13] claim that in its current form, the model is unbalanced in its levels of detail and too large to be practical. In the stratification to be introduced below, we review a more streamlined and extended account of information exchange levels.

4. Linguistic Levels

The definitions given in [11] agree in general, but differ substantially, with those used in the LCIM. They are summarized:

- *Pragmatics*: Data use in relation to data structure and context of application
- *Semantics*: Low level semantics focuses on definitions and attributes of terms; high level semantics focuses on the combined meaning of multiple terms (Generalized Context). Note in contrast to the LCIM requirement for semantic interoperability, this definition focuses on the underlying requirement for achieving shared meanings rather than how this requirement is achieved.
- *Syntax* focuses on a structure and adherence to the rules that govern that structure, e.g., XML (Rules and Structure)

Table 1 Levels of Conceptual Interoperability

Level of Conceptual Interoperability	**Characteristic**	**Key Condition**
Conceptual	The assumptions and constraints underlying the meaningful abstraction of reality are aligned	Requires that conceptual models be documented based on engineering methods enabling their interpretation and evaluation by other engineers.

Dynamic	Participants are able to comprehend changes in system state and assumptions and constraints that each is making over time, and are able to take advantage of those changes.	Requires common understanding of system dynamics
Pragmatic	Participants are aware of the methods and procedures that each is employing	Requires that the use of the data – or the context of their application – is understood by the participating systems.
Semantic	The meaning of the data is shared	Requires a common information exchange reference model
Syntactic	Introduces a common structure to exchange information,	Requires that a common data format is used
Technical	Data can be exchanged between participants	Requires that a communication protocol exists
Stand alone	No interoperability	

The authors of LCIM associate the lower layers with the problems of simulation interoperation while the upper layers relate to the problems of reuse and composition of models [14,15]. They conclude "simulation systems are based on models and their assumptions and constraints. If two simulation systems are combined, these assumptions and constraints must be aligned accordingly to ensure meaningful results."[10]. This suggests that levels of interoperability that have been identified in the area of modeling and simulation (M&S) can serve as guidelines to discussion of information exchange in general. Therefore, we consider an earlier developed conceptual layered architecture for M&S [16]. We'll correlate the above linguistic definitions with the layers outlined below and shown in Figure 2.

Network Layer contains the actual computers (including workstations and high performance systems) and the connecting networks (both LAN and WAN, their hardware and software) that do the work of supporting all aspects of the M&S lifecycle.

Execution Layer is the software that executes the models in simulation time and/or real time to generate their behavior. Included in this layer are the protocols that provide the basis for distributed simulation (such as those that are standardized in the High Level Architecture (HLA). Also included are database management systems, software systems to support control of simulation executions, visualization and animation of the generated behaviors.

Modeling Layer supports the development of models in formalisms that are independent of any given simulation layer implementation. HLA just mentioned also provides object-oriented templates for model description aimed at supporting confederations of globally dispersed models. However, beyond this, the formalisms for model behavior, whether continuous, discrete or discrete event in nature) as well as structure change, are also included in this layer. Model construction and especially, the key processes of model abstraction and continuity over the lifecycle are also included. We also add ontologies to this layer where they are understood as models of the world for a particular conceptualization intended to support information exchange.

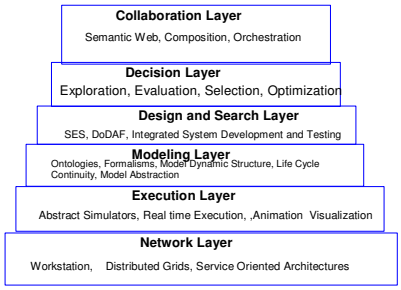

Figure 2 Architecture for Modeling and Simulation

Design and Search Layer supports the design of systems, such as in the Department of Defense Architecture Framework (DoDAF) where the design is based on specifying desired behaviors through models and implementing these behaviors through interconnection of system components. It also includes investigation of large families of alternative models, whether in the form of spaces set up by parameters or more powerful means of specifying alternative model structures such as provided by the SES methodology [11]. Artificial intelligence and simulated natural intelligence (evolutionary programming) may be brought in to help deal with combinatorial explosions

occasioned by powerful model synthesizing capabilities.

Decision Layer applies the capability to search and simulate large model sets at the layer below to make decisions in solving real-world problems. Included are course-of-action planning, selection of design alternatives and other choices where the outcomes may be supported by concept explorations, "what-if" investigations, and optimizations of the models constructed in the modeling layer using the simulation layer below it.

Collaboration Layer enables people or intelligent agents with partial knowledge about a system, whether based on discipline, location, task, or responsibility specialization, to bring to bear individual perspectives and contributions to achieve an overall goal.

Using the definitions for linguistic levels above, we correlate such levels with the layers just discussed. As illustrated in Figure 3, at the syntactic level we associate network and execution layers. The semantic level corresponds with the modeling layer – where we have included ontology frameworks as well as dynamic system formalisms as models. Finally, the pragmatic level includes use of the information such as identified in the upper layers of the M&S architecture. This use occurs for example, in design and search, making decisions and collaborating to achieve common goals. Indeed, such mental activities, along with real-world physical actions that they lead to, provide the basis for enumerating the kinds of pragmatic frames that might be of interest in particular applications – the context of use.

The resulting stratification leads us to propose Table 2 for defining effective interoperation of collaborating systems or services at the identified linguistic levels (first and second columns).

5. DEVS Standard

The conceptual interoperability model described above provides a general guideline for supporting system interoperability. Following the layered approach of this conceptual model, next we review the work of Discrete Event Systems Specification (DEVS) standardization that aims to support M&S interoperability based on the DEVS M&S framework. This work of standardization correspond to the two levels shown in Figure 3: the semantic level that deals with standardization of model interface; and the syntactic level that deals with standardization of simulation protocol.

The DEVS formalism [16], based on Mathematical Systems theory, provides a computational framework and tool set to support Systems concepts in application to SoS. We first provide a brief review. More detail is available in [16].

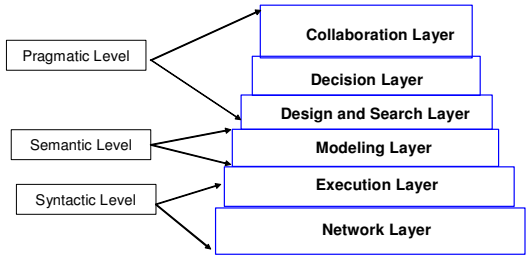

Figure 3 Associating Linguistic Levels with Layers of Modeling and Simulation

Table 2. Linguistic levels of Interoperability

Linguistic Level	**A collaboration of systems or services interoperates at this level if:**	**Examples**
Pragmatic – how information in messages is used	The receiver reacts to the message in a manner that the sender intends	An order from a commander is obeyed by the troops in the field as the commander intended. A necessary condition is that the information arrives in a timely manner and that its meaning has been preserved (semantic interoperability)
Semantic – shared understanding of meaning of messages	The receiver assigns the same meaning as the sender did to the message.	An order from a commander to multi-national participants in a coalition operation is understood in a common manner despite translation into different languages. Similarly geographic data must be translated correctly to UTM grid coordinates for ground

		forces and to LatLong for air and naval forces.
Syntactic – common rules governing composition and transmission of messages	The consumer is able to receive and parse the sender's message	A common network protocol (e.g. IPv4) is employed ensuring that all nodes on the network can send and receive data bit arrays adhering to a prescribed format.

DEVS makes a sharp distinction between the model and the device that simulates it. Both model and simulator are defined as mathematical systems as defined by Wymore and others (see [16] for details), and the relation between them is standardized by the concept of "abstract" simulator. Information flow in the DEVS formalism, as implemented on an object-oriented substrate, is mediated by the concept of DEVS message, a container for port-value pairs. In a message sent from component A to component B, a port-value pair is a pair in which the port is an output port of A, and the value is an instance of the base class of a DEVS implementation, or any of its sub-classes. A coupling is a four-tuple of the form (*sending component A, output port of A, receiving component B, input port of B*). This sets up a path where by a value placed on an output port of A by A's output function is transmitted to the input port of B, to be consumed by the latter. In systems or simulations implemented in DEVS environments the concepts of ports, messages, and coupling are explicit in the code. However, for systems/simulations that were implemented without systems theory guidance, in legacy or non-DEVS environments, these concepts are abstract and need to be identified concretely with the constructs offered by the underlying environment. For SoS engineering, where legacy components are the norm, it is worth starting with the clear concepts and methodology offered by systems theory and DEVS, getting a grip on the interoperability problems, and then translating backwards to the non-DEVS concepts as necessary.

Within a working group of the Simulation Interoperability 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 4 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 [www.acims.arizona.edu]. DEVS/C++ and DEVSJAVA are platform specific implementations while DEVSML[26] and FDDEVS [27] are platform independent implementations in XML which can transform to any platform specific implementations.

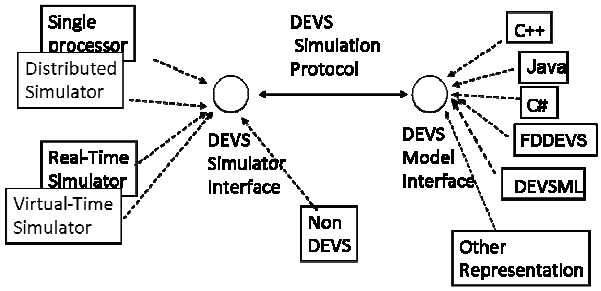

Figure 4: Conceptual Architecture of Standard

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. Corresponding to different simulation modes, the standard has virtual-time and real-time simulators. In virtual-time simulation, the simulator interprets time as logical time so the simulation can skip from one event time to the next without traversing the intervening time interval. However, in real-time simulation, time is interpreted as wall clock readings, so the real-time simulator will wait for the interval to its next scheduled event to expire before handling the event. In addition to the model type/simulation mode combinations, the standard allows for the use of different forms of distribution of model components, e.g., single processor vs. multi-processor, and within the latter, conservative vs optimistic time advance for virtual-time as well as centralized vs non-centralized time control in real-time execution. The standard is also agnostic with respect to different implementation platforms, such as Windows vs Unix, different programming languages, such as Java vs C++, and different networking and middleware frameworks such as .Net vs Apache. From the above introduction, we can see that the standard will have multiple simulation scenarios. For example, considering the combinations of simulation mode and distribution mode, we have: simulating a model in virtual-time and simulating model in real-time both in distributed and non-distributed fashion.

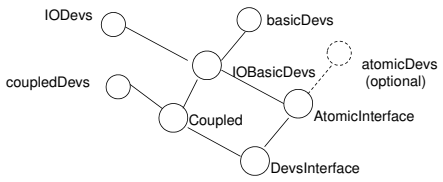

Figure 5 DEVS Model Interfaces

Among these interfaces, *IODevs* defines interface for the functions that handle message exchange based on input and output ports. Any model, whether DEVS or non-DEVS, can implement these functions so it can interoperate with other implementers of this interface, in the sense of receiving input and sending output. The *basicDevs* Interface defines the basic functions a DEVS model needs to implement such as *deltext()*, *deltint()*, *out()*, *ta()* and so on. The *basicDevs* interface is the interface that is exposed to the atomic simulators. An additional interface, *atomicDevs*, provides a convenient set of primitives for defining the basic functions in an atomic model. However, since the basic functions can be defined without using such primitives, the *atomicDevs* interface is optional. The *IOBasicDevs* interface extends the *IODevs* interface and *basicDevs* interface. It provides a common basis for implementing atomic models and coupled models. Combining *IOBasicDevs* with *atomicDevs*, we get *AtomicInterface* which defines the function signatures an atomic model need to implement. Of course, if *atomicDevs* is omitted, then *AtomicInterface* reduces to *IOBasicDevs*. Similarly, *CoupledDevs* interface defines the function signatures that are used in DEVS coupled models. It also has methods that support adding components and couplings to the model; methods for retrieving a component by name and for accessing all components; and to access the internal coupling specifications (intended only by simulators). Combining *IOBasicDevs* with *CoupledDevs*, we get the Coupled interface which defines the functions coupled models need to implement.

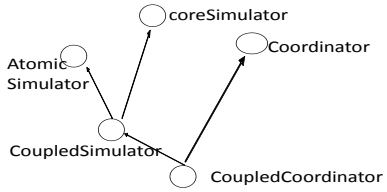

Figure 6 DEVS Simulator Interfaces

The basic simulator interface is the *CoreSimulator* that provides a common interface for DEVS and non-DEVS simulation. Further, the *CoreSimulator* interface is *the* basic interface from which simulation services could be designed for a truly net-centric interoperable simulation framework [23]. Under the *CoreSimulator* interface, two classes of simulators have been defined *CoupledSimulator* and *CoupledCoordinator* interfaces where the latter also inherits from *Coordintor*. These apply to both virtual (logical); and real-time simulation. (Real time simulators interpret time as real wall clock time and have their own thread and system clock. Virtual or logical time simulators can advance from one event time to the next). The *CoreSimulator* interface includes methods that are invoked by the DEVS simulation protocol:

```
interface coreSimulatorInterface{
void setSimulators
            (Collection<CoreSimulatorInterface>);
void initialize();
Double nextTN();
void computeInputOutput(Double t);
void applyDeltFunc(Double t);
void putContentOnSimulator
     (CoreSimulatorInterface sim, ContentInterface c);
void sendMessages();
```

5.1 DEVS Simulation Protocol

DEVS treats a model and its simulator as two distinct elements. The simulation protocol describes how a DEVS model should be simulated whether in standalone fashion or in a coupled model. Such a protocol is implemented by a processor which can be a simulator or a coordinator.

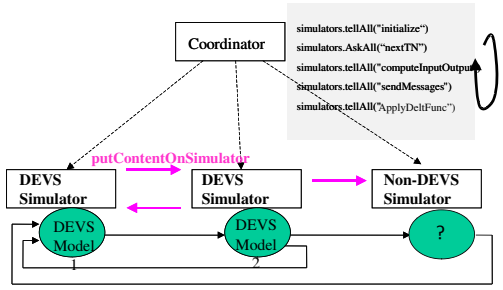

Figure 7 Federation of DEVS with Non-DEVS Simulators

As illustrated in Figure 7, the DEVS protocol is executed as following:

1. It starts with the coordinator telling each of the simulators in the collection the others' addresses and then to perform initialization function.
2. A cycle is then entered in which the coordinator requests that each simulator provide its time of next event and takes the minimum of the returned values to obtain the global time of next event
3. Each of the simulators applies its *computeInputOutput*() method to produce an output that consists of a collection of contents (port/value) pairs – for DEVS simulators this is a composite message computed according to the DEVS formalism based on its model's current state.
4. Then each simulator partitions its output into messages intended for recipient simulators and sends these messages to these recipient simulators – for DEVS simulators these recipients are determined from the output ports in the message and the coupling information that will have previously been received from the coordinator.
5. Finally, each simulator executes its *ApplyDeltFunc* method which computes the combined effect of the received messages and internal scheduling on its state, a side effect of which is produce of time of next event, *tN* – for DEVS simulators this state change is computed according to the DEVS formalism and the *tN* is updated using its model's time advance.
6. The coordinator obtains the next global time of next event and the cycle repeats

It should be noted that the above is one form of many possible protocols that can provide various forms of conservative and optimistic simulation, each of which must be proved to be correct as a realization of the DEVS closure under coupling property [16].

Implicit in the above description are the following constraints involving methods in the *CoreSimulatorInterface*:

- The *sendMessages*() method "must" employ the *putContentOnSimulator*() method as follows: for any simulator to which it wishes to send a content, it must call the recipient's *putContentOnSimulator*() method with the recipient and the content as arguments.
- Further, in applying its *computeInputOutput*() method, a simulator "must" be able to interpret the contents (satisfying the ContentInterface) it has received from the other simulators.

Notice that we cannot enforce the "must" requirements just given, and cannot prove that the simulation executes a desired behavior, unless we are given further information about its behavior. One way to do this is where the simulators are truly DEVS simulators in that they satisfy the interfaces and constraints given below. Failing this additional rigor, the interoperation involving DEVS and non-DEVS is purely at the technical level similar to that of a federation of simulators in HLA. This contrasts with the situation in which the federation is in fact derived from a DEVS coupled model for which correct simulation of the coupled model is guaranteed according to the DEVS formalism.

6. DEVS/SOA

An implementation of the standard within the Service Oriented Architecture (SOA) has been completed that provides DEVS modeling and simulation services over the World Wide Web [17, 23], As shown in the Figure 8, at the top of the layered architecture is the application layer that contains models in DEVSJAVA or DEVSML, a way of representing DEVS models in the eXtended Markup Language (XML). This DEVSML is built on JAVAML [18], which is 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 cross-transformable to Java. 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 SOA infrastructure and cam be executed at any remote

location in a distributed or non-distributed manner. Another major advantage of such capability is total simulator 'transparency'. The simulation engine is totally transparent to model execution over the SOA 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 schema [24] and a single coupled schema [25] 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 large coupled models using the DEVSML integration capabilities. Detailed design can be seen in [17,23].

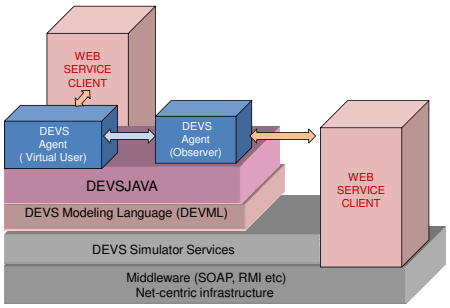

Figure 8 DEVS/SOA interoperability

The complete setup requires one or more servers that are capable of running DEVS Simulation Service, as shown in the second layer in Figure 8. The capability to run the simulation service is provided by the server side design of DEVS Simulation protocol supported by the DEVSJAVA. Of course, many issues of policy management and security considerations must be taken care of in the generation of DEVS models from WSDLs specifications [22]. Furthermore, the multi-platform simulation capability provided by DEVSV/SOA framework consists of realizing distributed simulation among different DEVS platforms or simulator engines such as DEVSJAVA, DEVS-C++, etc. and executing the native simulation service. This kind of interoperability where multi-platform simulations can be executed with our DEVSML integration facilities has been made possible with the hierarchical design of simulator interfaces as described in Section 5.

7. How Interoperability is supported

The proposed DEVS standard and its DEVS/SOA implementation support several modes of interoperability. These are outlined in the following paragraphs.

7.1 DEVS-to-DEVS Interoperability

DEVS-to-DEVS Interoperability is the basic form of interoperability enabled by the DEVS standard as discussed above. Adoption of the DEVS standard facilitates new development to achieve interoperability at the syntactic, semantic and pragmatic levels mentioned above. More detail on these concepts in application to testing of SOA systems can be found in [5, 20, 21, 22].

7.2 DEVS-to-Non-DEVS Interoperability

7.2.1 Direct. As mentioned before, legacy simulations that can be refactored to implement the *CoreSimulator* interface can be interoperate at the syntactic level with DEVS and other non-DEVS peers. In its strongest form, such simulation methodology guarantees well-defined time preservation and simulation correctness as a sound basis to aim for interoperability at the higher levels.

7.2.2 Via Client Gateways. For a variety of reasons, although DEVS compliance is desirable, it can be expected that legacy systems will continue to prevail and new non-compliant systems developed. The adoption of the SOA standard however, will facilitate the interoperation of DEVS and non-DEVS components that are compliant with the SOA standard. This form is realized in an Agent-implemented Test Instrumentation Infrastructure that deploys DEVS models to act as agents that are attached to clients of services [5,22]. Such attachment can be performed in automated fashion using tools such as Axis Toolkit to create the client stub given a service's Web Service Description Language (WSDL) [22]. As in Figure 8, these agents can observe the web service requests originating from the client and server responses (or failure thereof) to accumulate a variety of performance measurements. The agents can also serve as virtual users to interact with other users to direct the course of test scenarios and collect performance metrics to support scalability studies. Further, while collecting data, DEVS agents can communicate with each other to coordinate and share information using the DEVS-

to-DEVS configuration just discussed. Case studies are available in reference [22].

8. Conclusions

Achieving interoperability is one of the chief SoS engineering objectives in the development of command and control (C2) capabilities for joint and coalition warfare. The importance of M&S in SoS design and evaluation cannot be underestimated. M&S can be used strategically to provide early feasibility studies and aid the design process. As components comprising SoS are designed and analyzed, their integration and communication is the most critical part that must be addressed by the employed SoS M&S framework. The integration infrastructure must support interoperability at syntactic, semantic and pragmatic levels to enable such integration.

Currently there are several other approaches to distributed simulation and to integration of M&S with advanced C2 systems. These approaches build on the internet or other net-centric middleware to provide component connectivity and simulation services [1,20]. The latter may also include HLA implementations; however, the extent of adoption of HLA in this context remains to be seen. The DEVS standard provides a formal systems-based abstraction that can support higher level interoperability, whether alone or on top of HLA. The DEVS/SOA implementation provides a SOA implementation independent of HLA and is a viable approach to M&S integration with C2 SoS in the weaker gateway form, and in the strong direct compliance form. Further, DEVS has been applied to frameworks like DoDAF, UML and other systems engineering frameworks like SES. It is not a major step from here to see how DEVS components including decision making agents, sensor simulators, and environmental representations can bring the power of M&S to the development of C2 SoS. The underlying SOA standard that facilitates this interoperation can be expected to be widely adopted (for example, it has been adopted by the DoD's Global Information Grid initiative).

9. References

[1] Mark Pullen,LTC Ken Wilson, Michael Hieb, Andreas Tolk,Extensible Modeling and Simulation Framework (XMSF) C4I Testbed, http://www.movesinstitute.org/xmsf/xmsf.html

[2] Andrew Sage: From Engineering a System to Engineering an Integrated System Family, From Systems Engineering to System of Systems Engineering, 2007 IEEE International Conference on System of Systems Engineering (SoSE). April 16th -18th, 2007, San Antonio, Texas

[3] Jacobs, Robert W. "Model-Driven Development of Command and Control Capabilities For Joint and Coalition Warfare," Command and Control Research and Technology Symposium, June 2004.

[4] Dahmann, J.S., F. Kuhl, and R. Weatherly, Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture For Simulation. Simulation, 1998. 71(6): p. 378

[5] Saurabh Mittal, Bernard P. Zeigler, Jose L. Risco Martin, Ferat Sahin and Mo Jamshidi Modeling and Simulation for Systems of Systems Engineering to appear in Systems of Systems -- Innovations for the 21st Century (to be published by Wiley)

[6] Tolk, A., and Muguira, J.A. The Levels of Conceptual Interoperability Model (LCIM). Proceedings Fall Simulation Interoperability Workshop, 2003

[7] M.J. DiMario System of Systems Interoperability Types and Characteristics in Joint Command and Control, Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering, Los Angeles, CA, USA - April 2006

[8] Levels of Information Systems Interoperability (LISI), http://www.sei.cmu.edu/isis/guide/introduction/lisi.htm

[9] Turnitsa C., and A. Tolk, "Evaluation of the C2IEDM as an Interoperability-Enabling Ontology," Proceedings of Fall Simulation Interoperability Workshop, 2005.

[10] Muguira, James. and Tolk., A "Applying a Methodology to identify Structural Variances in Interoperations," JDMS: The Journal of Defense Modeling and Simulation, Vol 3, No 2, 2006

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

[12] Turnitsa C., and A. Tolk, "Evaluation of the C2IEDM as an Interoperability-Enabling Ontology," Proceedings of Fall Simulation Interoperability Workshop, 2005.

[13] Lasschuyt E., M. van Henken, W. Treurniet, and M. Visser, "How to Make an Effective Information Exchange Data Model," RTO-IST-042/9,2004

[14] Hoffmann, M., Challenges of Model Interoperation in Military Simulations. SIMULATION, Vol. 80, pp. 659-667, 2004

[15] Chaum, E., Hieb, M.R., and Tolk, A. "M&S and the Global Information Grid," Proceedings Interservice/Industry Training, Simulation and Education Conference (I/ITSEC), 2005.

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

[17] Mittal, S., Risco-Martin, J.L., Zeigler, B.P. "DEVS-Based Web Services for Net-centric T&E", Summer Computer Simulation Conference, 2007

[18] Badros, G. JavaML: a Markup Language for Java Source Code, Proceedings of the 9th International World Wide Web Conference on Computer Networks: the international journal of computer and telecommunication networking, pages 159-177

[19] Zeigler, B. P., S Mittal, "Enhancing DoDAF with DEVS-Based System Life-cycle Process", IEEE International Conference on Systems, Man and Cybernetics, Hawaii, October 2005

[20] Steven W. Reichenthal, SRML - Simulation Reference Markup Language W3C Note 18 December 2002 http://www.w3.org/TR/SRML/

[21] S Mittal, "Extending DoDAF to allow DEVS-Based Modeling and Simulation", Special issue on DoDAF, Journal of Defense Modeling and Simulation (JDMS), Vol 3. No. 2

[22] S Mittal, BP Zeigler, JLR Martin, J Nutaro, "Design and Analysis of Service Oriented Architectures using DEVS/SOA-Based Modeling and Simulation", submitted to IEEE Transactions on Systems, Man and Cybernetics, Part C, Special Issue on Information Reuse and Integration

[23] S. Mittal, JLR Martin, BP Zeigler, "DEVS/SOA: A Cross-platform Framework for Net-centric Modeling and Simulation in DEVS Unified Process", submitted to SIMULATION: Transactions of SCS
[24] Atomic Schema: http://www.u.arizona.edu/~saurabh/fddevs/NewXMLSchema.xsd
[25] Coupled Schema: http://www.u.arizona.edu/~saurabh/fddevs/CoupledDevs.xsd
[26]S Mittal, JLR Martín, BP Zeigler, "*DEVSML: Automating DEVS Execution over SOA Towards Transparent Simulators*", Special Session on DEVS Collaborative Execution and Systems Modeling over SOA, DEVS Integrative M&S Symposium DEVS' 07, Spring Simulation Multi-Conference, March 2007
[27] S Mittal, MH Hwang, BP Zeigler, Finite Deterministic DEVS (FDDEVS): http://www.u.arizona.edu/%7Esaurabh/fddevs/FD-DEVS.html