System of Systems – Innovations for the 21st Center, Edited by [Mo Jamshidi].
ISBN 0-471-XXXXX-X

5 Modeling and Simulation for Systems of Systems Engineering

Saurabh Mittal, Ph.D.

Bernard P. Zeigler, Ph.D.

Arizona Center for Integrative Modeling and Simulation
Electrical and Computer Engineering,
University of Arizona
Tucson, AZ

José L. Risco Martín, Ph.D.

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid
Madrid, Spain

Ferat Sahin, Ph.D.

Multi Agent Bio-Robotics Laboratory
Electrical Engineering
Rochester Institute of Technology
Rochester, NY

Mo Jamshidi, Ph.D.

Lutcher Brown Endowed Chair
Electrical and Computer Engineering
University of Texas San Antonio
San Antonio, TX

Abstract: A critical aspect and differentiator of a System of Systems (SoS) versus a single monolithic system is interoperability among the constituent disparate systems. A major application of Modeling and Simulation (M&S) to SoS Engineering is to facilitate system integration in a manner that helps to cope with such interoperability problems. A case in point is the integration infrastructure offered by the DoD Global Information Grid (GIG) and its Service Oriented Architecture (SOA). In this chapter, we discuss a process called DEVS Unified Process (DUNIP) that uses the Discrete Event System Specification (DEVS) formalism as a basis for integrated system engineering and testing called the Bifurcated Model-Continuity life-cycle development methodology. DUNIP uses an XML-based DEVS Modeling Language (DEVSML) framework that provides the capability to compose models that may be expressed in a variety of DEVS implementation languages. The models are deployable for remote and distributed real-time executing agents over the Service Oriented Architecture (SOA) middleware. In this paper, we formulate a methodology for testing any proposed SOA-based integration infrastructure, such as DISA's Net-Centric Enterprise Services. To support such a methodology we use DUNIP to define a Test Instrumentation System (TIS) that employs the DEVS/SOA infrastructure to deploy agents to test the mission success and performance levels of collaborations over the GIG/SOA.

1. Key words: DEVS, DEVS/SOA, DEVSML, DUNIP, Bifurcated Model-Continuity Based Life-Cycle Methodology

1. INTRODUCTION

The System of Systems (SoS) concept, originally suggested as a method to describe the use of different systems interconnected to achieve a specific goal, has grown in its myriad of definitions and concepts [DIM07,DIM06]. Nevertheless, a common defining attribute of a SoS that critically differentiates it from a single monolithic system is interoperability, or lack thereof, among the constituent disparate systems. The plethora of perspectives on SoS problems evident in the literature [SAG07, MOR04, SAG01] suggest that interoperability may take the form of integration of constituent systems (e.g., element A is hierarchically superior to element B) or interoperation of constituent systems (e.g. two or more independent elements or systems with no identified hierarchy). In this chapter we focus less on SoS problems per se than on the role that M&S can play in helping to address these problems.

Systems theory, especially as formulated by Wymore [WAY92, WAY62], provides a conceptual basis for formulating the interoperability problem of SoS. Systems are viewed as components to be coupled together to form a higher level system, the SoS. As illustrated in Exhibit 1, components have input and output ports (indicated with arrows) that allow couplings (indicated in dashed connectors) to be defined through which information can flow from output ports to input ports. The DEVS formalism [ZEI00], based on Systems theory, provides a computational framework and tool set to support Systems concepts in application to SoS.

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, in zero time, to the input port of B, to be consumed by the latter's external transition function[1].

[1] The confluent function may also be involved; see [ZEI00] for details

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.

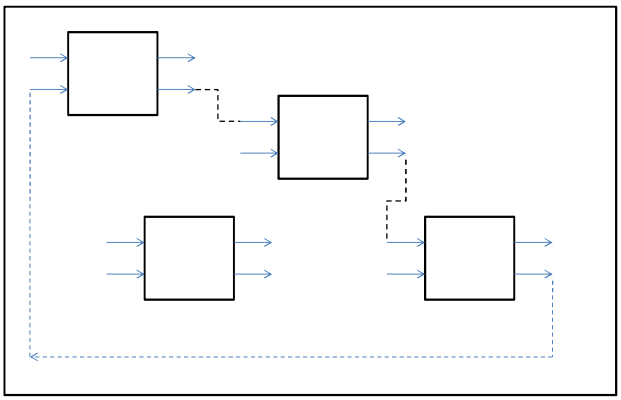

Exhibit 1: System composite of components systems or coupled model

1.1 Levels of Interoperability

With the conceptual basis offered by the DEVS formalism in mind, we briefly review experience with interoperability in the distributed simulation context and a linguistically based approach to the SoS interoperability problem [DIM06,CAR05]. Sage [SAG07] drew the parallel between viewing the construction of SoS as federation of systems and the federation that is supported by the High Level Architecture (HLA), an IEEE standard fostered by the DoD to enable composition of simulations [SAR00,DAH98]. As illustrated in Exhibit 2, HLA is a network middleware layer that supports message exchanges among simulations, called federates, in a neutral format[2]. However, experience with HLA has been disappointing and forced acknowledging the difference between enabling heterogeneous simulations to exchange data (so-called technical interoperability) and the desired outcome of exchanging meaningful data so that coherent interaction among federates takes place, so-called, substantive interoperability [YLMAZ]. Tolk introduced the Levels of Conceptual Interoperability Model (LCIM) which identified seven levels of interoperability among participating systems [TOL03]. These levels can be viewed as a refinement of the *operational* interoperability type which is one of three defined by Dimario [DIM06]. The operational type concerns linkages between systems in their interactions with one another, the environment, and with users. The other types apply to the

[2] HLA also provides a range of services to support execution of simulations

context in which systems are constructed and acquired. They are *constructive* – relating to linkages between organizations responsible for system construction and *programmatic* – linkages between program offices to manage system acquisition.

Subsequently, one of the present authors co-authored a book in which the LCIM was mapped into three linguistically-inspired levels: *syntactic, semantic,* and *pragmatic.* The levels are summarized in Exhibit 3. More detail is provided in [ZEI07].

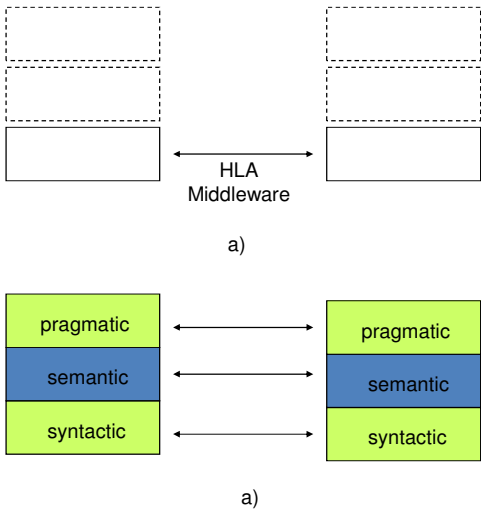

Exhibit 2: Interoperability levels in distributed simulation

Linguistic Level	**A collaboration of systems or services interoperates at this level if:**
Pragmatic – how information in messages is used	The receiver re-acts to the message in a manner that the sender intends (assuming non-hostility in the collaboration).
Semantic – shared understanding of meaning of messages	The receiver assigns the same meaning as the sender did to the message.
Syntactic – common rules governing composition and transmitting of messages	The consumer is able to receive and parse the sender's message

Exhibit 3: Linguistic levels

In this interoperability framework, the question to be addressed here is how M&S can help to achieve all three linguistic levels of interoperability. To discuss this question in more depth, we proceed to a brief review of the formal foundations of M&S that will allow us to frame the question and discuss the support available now and possible in the future.

2. REVIEW OF M&S FOUNDATIONAL FRAMEWORK

The theory of modeling and simulation presented in [ZEI00] provides a conceptual framework and an associated computational approach to methodological problems in M&S. The framework provides a set of entities (real system, model, simulator, experimental frame) and relations among the entities (model validity, simulator correctness, among others) that, in effect, present a ontology of the M&S domain. The computational approach is based on the mathematical theory of systems and works with object orientation and other computational paradigms. It is intended to provide a sound means to manipulate the framework elements and to derive logical relationships among them that are usefully applied to real world problems in simulation modeling. The framework entities are formulated in terms of the system specifications provided by systems theory, and the framework relations are formulated in terms of the morphisms (preservation relations) among system specifications. Conversely, the abstractions provided by mathematical systems theory require interpretation, as provided by the framework, to be applicable to real world problems.

In its computational realization, the framework is based on the Discrete Event System Specification (DEVS) formalism and implemented in various object oriented environments. Using Unified Modeling Language (UML) we can represent the framework as a set of classes and relations as illustrated in Exhibits 4 and 5.

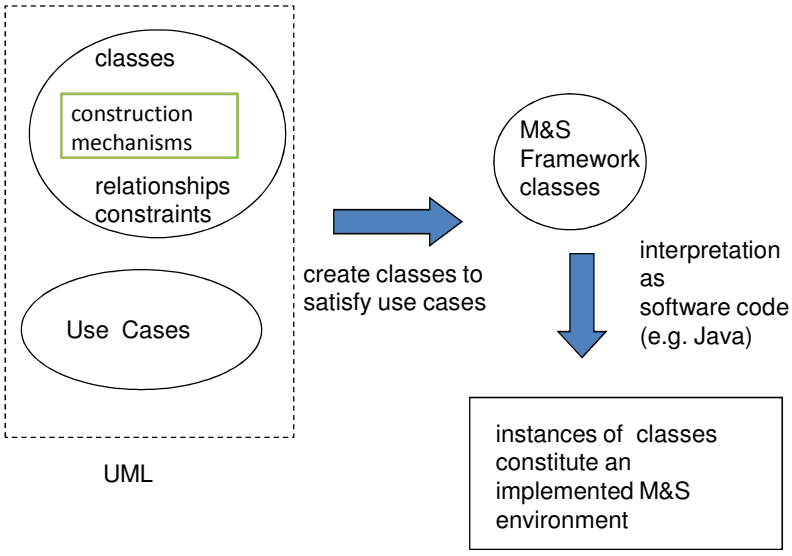

Exhibit 4: M&S Framework formulated within UML

Various implementations support different subsets of the classes and relations [OMG]. In particular, this chapter will review the implementation of DEVS within a Service Oriented Architecture (SOA) environment called DEVS/SOA [MIT07g, DUN07, MIT07f].

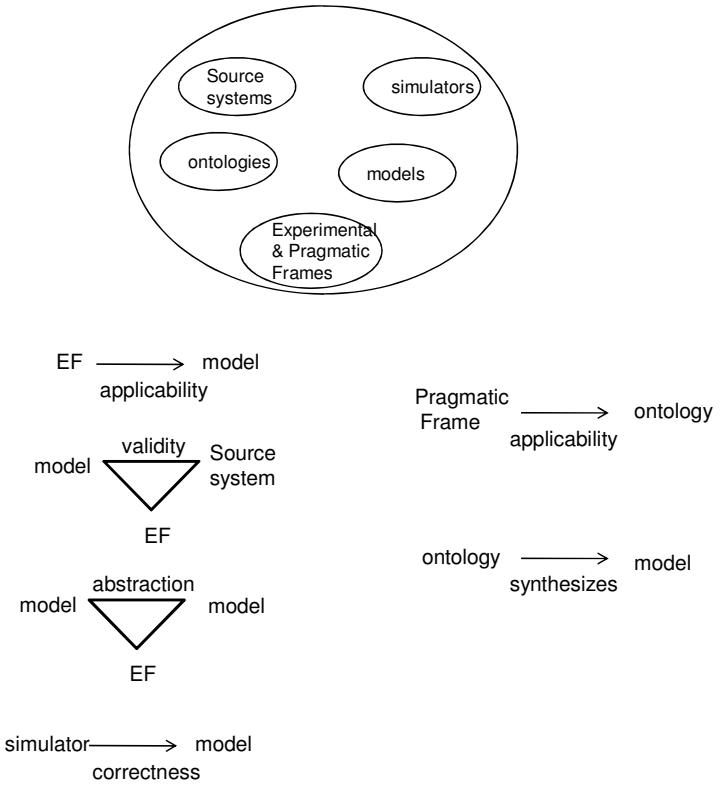

Exhibit 5: M&S Framework Classes and Relations in a UML representation

In a System of systems, systems and/or subsystems often interact with each other because of interoperability and over all integration of the SoS. These interactions are achieved by efficient communication among the systems using either peer-to-peer communication or through central coordinator in a given SoS. Since the systems within SoS are operationally independent, interactions among systems are generally asynchronous in nature. A simple yet robust solution to handle such asynchronous interactions (specifically, receiving messages) is to throw an event at the receiving end to capture the messages from single or multiple systems. Such system interactions can be represented effectively as discrete-event models. In discrete-event modeling, events are generated at random time intervals as opposed to some pre-determined time interval seen commonly in discrete-time systems. More specifically, the state change of a discrete-event system happens only upon arrival (or generation) of an event, not necessarily at equally spaced time intervals. To this end, a discrete-event model is a feasible approach in simulating the SoS framework and its interaction. Several discrete-event simulation engines [xMAT, xOMN, xNS2, xDEVS] are available that can be used in simulating interaction in a heterogeneous mixture of independent systems. The advantage of DEVS is its effective mathematical representation and its support to distributed simulation using middleware such as DoD's High Level Architecture (HLA) [xHLA].

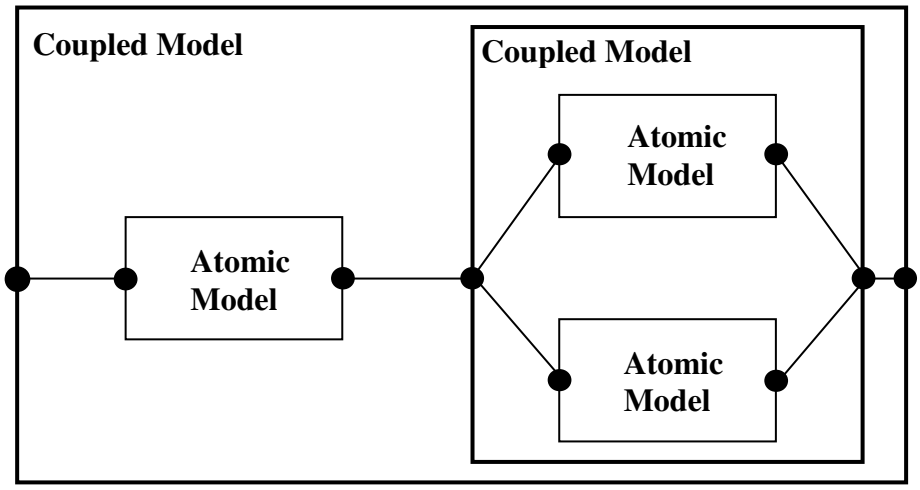

Exhibit 6: DEVS Hierarchical Model representation for systems and sub-systems

2.1.1 DEVS Modeling and Simulation

Discrete Event System Specification (DEVS) [ZEI00] is a formalism, which provides a means of specifying the components of a system in a discrete event simulation. In DEVS formalism, one must specify *Basic Models* and how these models are connected together. These basic models are called *Atomic Models* and larger models which are obtained by connecting these atomic blocks in meaningful fashion are called *Coupled Models* (shown Exhibit 6). Each of these atomic models has *inports* (to receive external events), *outports* (to send events), set of *state variables*, *internal transition*, *external transition*, and *time advance functions*. Mathematically it is represented as 7-tuple system: $M =< X, S, Y, \delta_{\text{int}}, \delta_{ext}, \lambda, t_a >$ where *X* is an input set, *S* is set of states, *Y* is set of outputs, δ_{int} is internal transition function, δ_{ext} is external transition function, λ is the output function, and t_a is the time advance function. The model's description (implementation) uses (or discards) the message in the event to do the computation and delivers an output message on the outport and makes a state transition. A Java-based implementation of DEVS formalism, DEVSJAVA [SAR00]**,** can be used to implement these atomic or coupled models. In addition, DEVS-HLA [SAR00] will be helpful in distributed simulation for simulating multiple heterogeneous systems in the System of systems framework.

The following section explores how XML and DEVS environment can be combined in a simulation environment.

2.1.2 XML and DEVS

In DEVS, messages can be passed from one system (coupled or atomic model) to another using either predefined or user-defined message formats. Since the systems within SoS maybe different in hardware and/or software, there is a need for a unifying language for message passing. Each system need not necessarily have the knowledge (operation, implementation, timing, data issues, and so on) of another system in a SoS. Therefore, one has to work at a high-level (information or data level) in order to understand the present working condition of the system. One such good fit for representing different data in a universal manner is XML. Exhibit 7 describes conceptually a SoS simulation example to demonstrate the use of XML as a message passing paradigm using DEVS formalism.

In Exhibit 7, there are three systems in a hierarchy where systems A and B send and receive data from system C. System C sends and receives data from a higher level as described in the message of system C. The data sent by system C has data from systems A and B. In addition, it has information about system A and B being system C's subsystems.

Assuming the XML based integration architecture and the DEVS environment exist, we can then offer solutions to robust data aggregation and fusion for system of systems which have heterogeneous complex systems such as mobile sensor platforms or micro-devices.

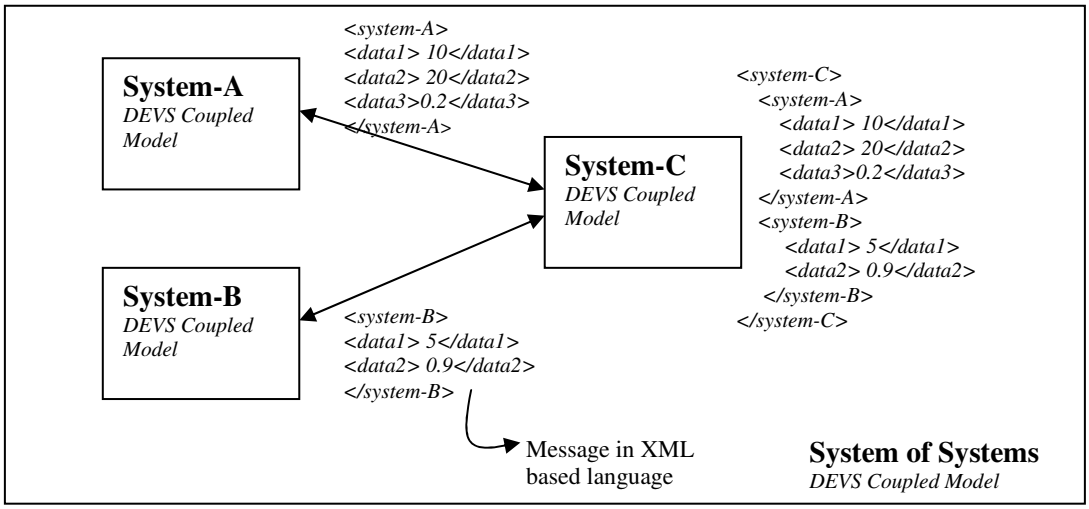

Exhibit 7: A SoS simulation example with three systems and XML-like message passing [SAH07a].

3. MODEL BASED ENGINEERING

Model-based Software Engineering process is commonly referred as Model Driven Architecture (MDA) or Model-Driven Engineering. The basic idea behind this approach is to develop model before the actual artifact or product is designed and then transform the model itself to the actual product. The MDA is pushed forward by Object Management Group (OMG) since 2001. The MDA approach defines system functionality using platform-independent model (PIM) using an appropriate domain-specific language. Then given a Platform Definition Model (PDM), the PIM is translated to one or more platform-specific models (PSMs). The OMG documents the overall process in a document called MDA guide.

MDA is a collection of various standards like the Unified Modeling Language (UML), the Meta-Object Facility (MOF), the XML Metadata Interchange (XMI), Common Warehouse Model (CWM) and a couple of others. OMG focuses Model-driven architecture on forward engineering i.e. producing code from abstract, human-elaborated specifications [Wikipedia].

An MDA tool is used to develop, interpret, compare, align etc. models or meta-models. A 'model' is interpreted as meaning any kind of models (e.g. a UML model) or metamodel (e.g. CWM metamodel). An MDA tool may be one or more of the following types:

- Creation tool: Used to elicit initial models and /or edit derived models
- Analysis tool: Used to check models for completeness, inconsistencies or define any model metrics

- Transformation tool: Used to transform models into other models or into code and documentation
- Composition tool: Used to compose several source models, preferably conforming to the same metamodel
- Test tool: Used to "test" models. A mechanism in which test cases are derived in whole or in part from a model that describes some aspects of system under test (SUT)
- Simulation tool: Used to simulate the execution of system represented by a given model. Simply speaking, is the mechanism by which model is 'executed' using a programming language
- Reverse Engineering tool: Intended to transform a particular legacy or information artifact into full-fledged models.

It is not required that one tool may contain all of the features needed for Model Driven Engineering. UML is a small subset of much broader scope of UML. Being a subset of MDA, the UML is bounded by its own UML metamodel. Progress has been made to develop executable UML models but it has not gained industry wide mainstream acceptance for the same limited scope. Potential concerns with the current MDA state of art include:

- MDA approach is underpinned by a variety of technical standards, some of which are yet to be specified (e.g. executable UML)
- Tools developed my many vendors are not interoperable
- MDA approach is considered too-idealistic lacking iterative nature of Software Engineering process
- MDA practice requires skilled practitioners and design requires engineering discipline not commonly available to code developers.
- OMG sponsored CORBA project after much promises but it failed to materialize as a widely accepted standard.

Model-based Testing is a variant of testing that relies on explicit behavior models that encode the intended behavior of the system and possibly the behavior of its environment [Utt06]. Pairs of input and output of the model of the implementation are interpreted as test-cases for this implementation: the output of the model is the expected output of the system under test (SUT). This testing methodology must take into account the involved abstractions and the design issues that deals with lumping different aspects as these can not be tested individually using the developed model.

Following is the process for Model-based testing technique [Utt06] as shown in Exhibit 8:

1. a model of the SUT is built on existing requirements specification with desired abstraction levels
2. Test selection criteria are defined with an objective to detect severe and likely faults at an acceptable cost. These criteria informally describe the guidelines for a test suite.
3. Test selection criteria are then translated into test case specifications. It is an activity where a textual document is turned 'operational'. Automatic test case generators fall into this step of execution.

4. A test suite is 'generated' that is built upon the underlying model and test case specifications.

5. Test cases from the generated test suite are run on the SUT after suitable prioritization and selection mechanism. Each run results in a verdict of 'passed' or 'failed' or 'inconclusive'.

A summary of contributions to the Model-based Testing domain can be seen at [Utt06].

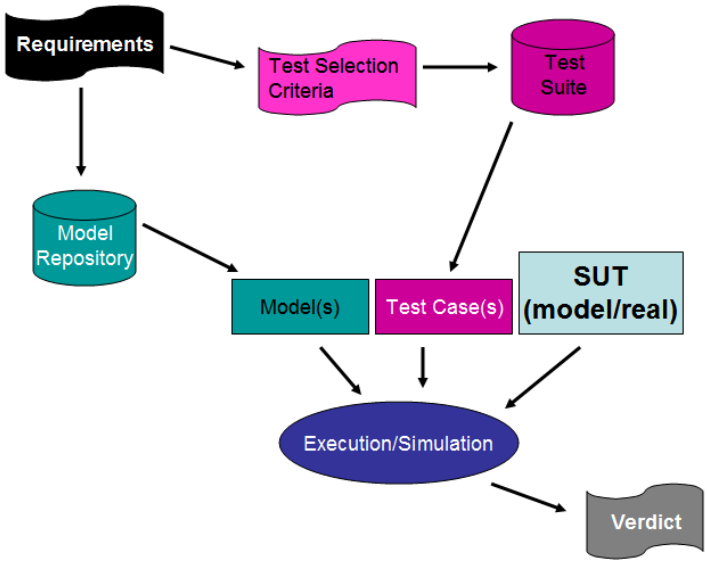

Exhibit 8: Graphical process extended further from [Utt06]

4. SOS ARCHITECTURE MODELING: DODAF, UML AND SYSTEMS ENGINEERING PRINCIPLES

Unified Modeling Language (UML) [UML]has been widely adopted by the industry as the preferred means of architecture specification due to its multi-model expressive power. However, UML constructs are not sufficient to specify the complete set of SoSE processes. A more extensive architectural framework is needed for better organization and management of SoSE Artifacts. Frameworks such as Wymore's [WAY92, WAY62], the Department of Defense Architecture Framework (DoDAF) [DOD03a], and Zachman's [ZACH], are examples that may use UML as a means of presenting the concepts of SoSE. There are other representation mechanisms like IDEF [IDEF] notation that help understand various SE perspectives. However, UML is more comprehensive in its graphical representations and these can aid SoSE frameworks in their descriptions of various portions of SE processes.

Exhibit 9 represents the integral role that M&S plays in SoSE processes with respect to Wymore's theory for systems engineering. DEVS is strategically placed between the I/O requirements and the technology requirements to provide M&S at 'design' level, before a design is deemed 'feasible'. DEVS is based on 'port' identification and is component-based modeling and simulation formalism that meets at the crossroads of the two cotyledons of Wymore's theory.

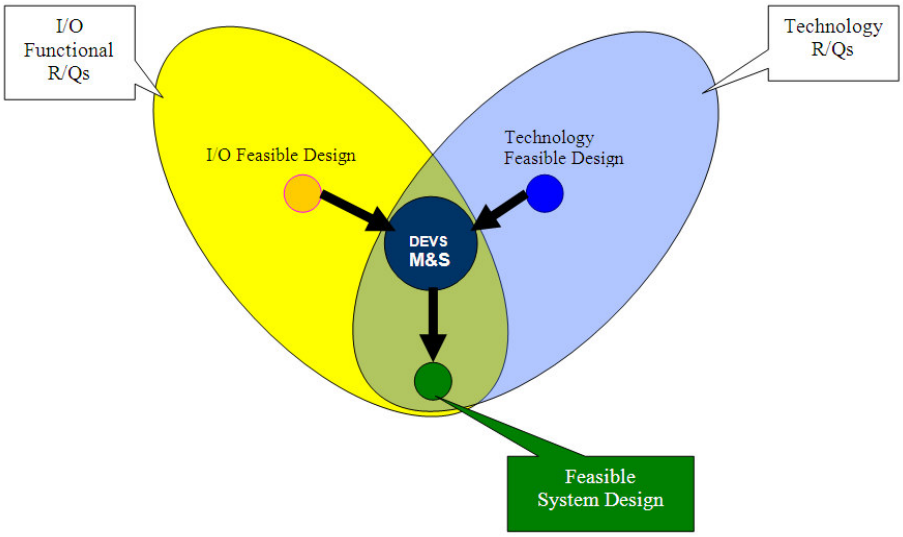

Exhibit 9: Role of M&S in Wymorian tri-cotyledon theory for Systems engineering

Wymore's Theory of Systems Engineering also underlies our system-theory and DEVS-based approach to SoSE, viz., integrated M&S for Testing and Evaluation strategy as described in sections ahead.

DoDAF is the basis for the integrated architectures mandated in DOD Instruction 5000.2 [DOD03b] and provides broad levels of specification related to operational, system, and technical views. Integrated architectures are the foundation for interoperability in the joint Capabilities Integration and Development System (JCIDS) prescribed in CJCSI 3170.01D and further described in CJCSI 6212.01D [CJC04,CJC06]. DoDAF seeks to overcome the plethora of "stove-piped" design models that have emerged. Integration of such legacy models is necessary for two reasons. One is that, as systems, families of systems, and systems-of-systems become more broad and heterogeneous in their capabilities, the problems of integrating design models developed in languages with different syntax and semantics has become a serious bottleneck to progress. The second is that another recent DoD mandate also intended to break down this "stove-piped" culture requires the adoption of the Service Oriented Architecture (SOA) paradigm as supported in the development of Network Centric Enterprise Services (NCES) [DOD05]. However, anecdotal evidence suggests that a major revision of the DoDAF to support net-centricity is widely considered to be needed and efforts are underway towards a SOA based DoDAF.

DoDAF consists of three views, viz., Operational View (OV), Systems View (SV), and Technical View (TV). OV is a description of the tasks and activities, operational elements, and information exchanges required to accomplish DoD missions. DoD mission include both the warfighting missions and business process. These are further decomposed into separate mission threads. SV is a set of graphical and textual products that describes systems and interconnections providing for, or supporting, DoD functions. SV associates system resources to the requirements of OV. TV is the minimal set of rules governing the arrangement, interaction, and interdependence of system parts or elements, whose purpose is to ensure that a conformant system satisfies a specified set of requirements.

M&S is introduced at the intersection of the modeling and technology cotyledons in Wymore's theory (recall Exhibit 9) to help bridge the gap between the desired and the possible. Similarly, we can augment DoDAF by introducing M&S at a place where the design goes from abstraction to realism i.e., from Operational View (OV) specifications to System View (SV) implementations. Exhibit 10 depicts the fact that M&S can contribute to the system design process as well.

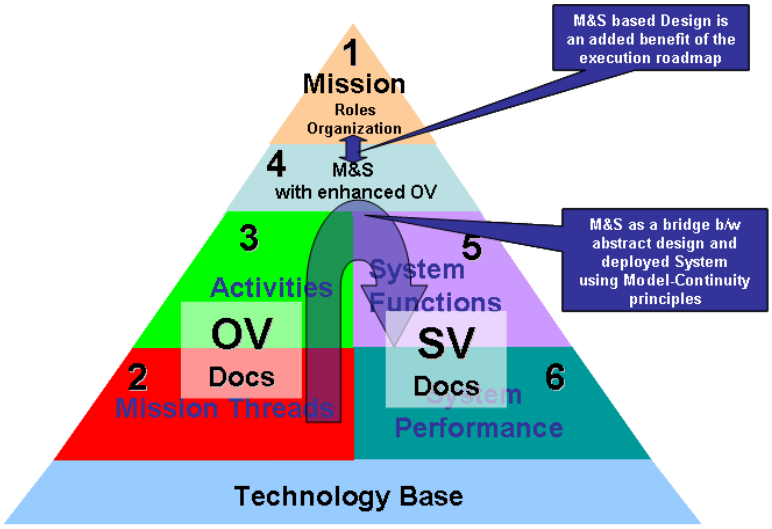

Exhibit 10: Role of M&S in DoDAF design process

Although the current DoDAF specification provides an extensive methodology for system architectural development, it is deficient in several related dimensions – absence of integrated modeling and simulation support, especially for model-continuity throughout the development process, and lack of associated testing support. To overcome these deficiencies, we described an approach to support specification of DoDAF architectures within a development environment based on DEVS-based modeling and simulation. The authors [MIT06a,ZEI05b] enhanced the DoDAF specification to incorporate M&S as a means to develop 'executable architecture' [ATK04] from DoDAF specifications and provided detailed DoDAF to DEVS mapping leading to simulation, and feasibility analysis. The result is an enhanced system lifecycle development process that includes model-continuity based development and testing in an integral manner.

5. SYSTEMS OF SYSTEMS TEST AND EVALUATION USING DEVS M&S

5.1.1 DEVS State-of-the-art in Test and Evaluation

Before moving onto the details of DEVS Unified Process (DUNIP), it is important to list the capabilities of DEVS technology as it stands today and how it contributes to the system of Systems test and evaluation. Further, such test and evaluation (T&E) must be viewed within SOA perspective due to various DoD mandates. The following Exhibit 11 summarizes DEVS state-of-the-art with respect to T&E.

Desired M&S Capability for T&E	**Solutions Provided by DEVS Technology**
Support of DoDAF need for executable architectures using M&S such as mission based testing for GIG SOA	DEVS Unified Process [MIT07g] provides methodology and SOA infrastructure for integrated development and testing, extending DoDAF views [MIT06a].
Interoperability and cross-platform M&S using GIG/SOA	Simulation architecture is layered to accomplish the technology migration or run different technological scenarios [SAR01, MIT03d]. Provide net-centric composition and integration of DEVS 'validated' models using Simulation Web Services [MIT07e]
Automated test generation and deployment in distributed simulation	Separate a model from the act of simulation itself, which can be executed on single or multiple distributed platforms [ZEI00]. With its bifurcated test and development process, automated test generation is integral to this methodology [ZEI05].
Test artifact continuity and traceability through phases of system development	Provide rapid means of deployment using model-continuity principles and concepts like "simulation becomes the reality" [HUX03].
Real time observation and control of test environment	Provide dynamic variable-structure component modeling to enable control and reconfiguration of simulation on the fly [MIT06b,MIT05c,MIT03d, HUX03,]. Provide dynamic simulation tuning, interoperability testing and benchmarking.

Exhibit 11: DEVS State-of-the-art

In an editorial [CAR05], Carstairs asserts an acute need for a new testing paradigm that could provide answers to several challenges described in a three-tier structure. The lowest level, containing the individual systems or programs, does not present a problem. The second tier, consisting of systems of systems in which interoperability is critical, has not been addressed in a systematic manner. The third tier, the enterprise level, where joint and coalition operations are conducted, is even more problematic. Although current test and evaluation (T&E) systems are approaching adequacy for tier-two challenges, they are not sufficiently well integrated with defined architectures focusing on interoperability to meet those of tier three. To address mission thread testing at the second and third tiers, Carstairs advocates a collaborative distributed environment (CDE), which is a federation of new and existing facilities from commercial, military, and not-for-profit organizations. In such an environment, modeling and simulation (M&S) technologies can be exploited to support model-continuity [Hux04] and model-driven design (MDD) development [Weg02], making test and evaluation an integral part of the design and operations life-cycle.

The development of such a distributed testing environment would have to comply with recent Department of Defense (DoD) mandates requiring that the DoD Architectural Framework (DoDAF) be adopted to express high-level system and operational requirements and architectures

[DOD03a, DOD03b, CJC04, CJC06]. Unfortunately, DoDAF and DoD net-centric [ATK04] mandates pose significant challenges to testing and evaluation since DoDAF specifications must be evaluated to see if they meet requirements and objectives, yet they are not expressed in a form that is amenable to such evaluation. DoDAF does not provide a formal algorithmically-enabled process to support such integration at higher resolutions. Lacking such processes, DoDAF is inapplicable to the SOA domain and GIG in particular. There have been efforts like [DAN04] that have tried to map DoDAF products to SOA but as it stands out there is no clear-cut methodology to develop an SOA directly from DoDAF, rest aside their testing and evaluation.

5.2 Bifurcated Model-Continuity Based Life-Cycle Methodology

The needed solution is provided by combining the systems theory, M&S framework and model-continuity concepts that lead naturally to a formulation of a Bifurcated Model-Continuity based Life-cycle process as illustrated in Exhibit 12. The process can be applied to development of systems using model-based design principles from scratch or as a process of reverse engineering in which requirements have already been developed in an informal manner. The depicted process is a universal process and is applicable in multiple domains. The objective of this research effort is to incorporate DEVS as the binding factor at all phases of this universal process.

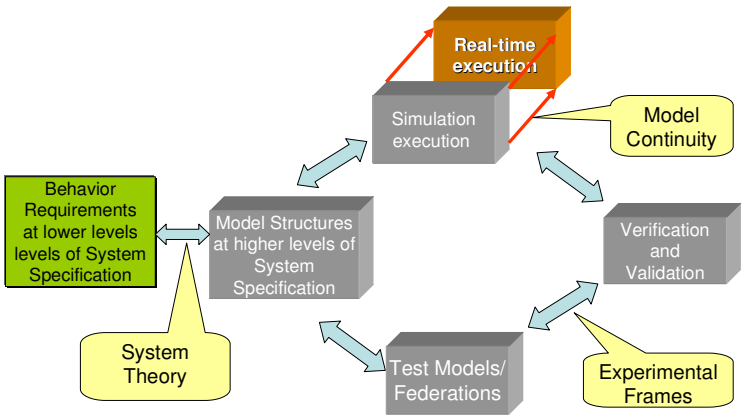

Exhibit 12: Bifurcated Model-Continuity based System Life-cycle Process

The process has the following characteristics:

- **Behavior Requirements at lower levels of System Specification**: The hierarchy of system specification as laid out in [Zeig00] offers well-characterized levels at which requirements for system behavior can be stated. The process is essentially iterative and leads to increasingly rigorous formulation resulting from the formalization in subsequent phases.

- **Model Structures at higher levels of System Specification:** The formalized behavior requirements are then transformed to the chosen model implementations e.g. DEVS based transformation in C++, Java, C# and others.
- **Simulation Execution:** The model base which may be stored in Model Repository is fed to the simulation engine. It is important to state the fact that separating the Model from the underlying Simulator is necessary to allow independent development of each. Many legacy systems have both the Model and the Simulator tightly coupled to each other which restrict their evolution. DEVS categorically separates the Model from the Simulator for the same simple reason.
- **Real-time Execution:** The simulation can be made executable in real-time mode and in conjunction with Model-Continuity principles, the model itself becomes the deployed code
- **Test Models/Federations:** Branching in the lower-path of the Bifurcated process, the formalized models give way to test models which can be developed at the atomic level or at the coupled level where they become federations. It also leads to the development of experiments and test cases required to test the system specifications. DEVS categorically aids the development of Experimental Frames at this step of development of test-suite.
- **Verification and Validation:** The simulation provides the basis for correct implementation of the system specifications over a wide range of execution platforms and the test suite provides basis for testing such implementations in a suitable test infrastructure. Both of these phases of systems engineering come together in the Verification and Validation (V&V) phase.

6. EXPERIMENTAL FRAME CONCEPTS

An experimental frame is a specification of the conditions under which the system is observed or experimented with. As such an experimental frame is the operational formulation of the objectives that motivate a modeling and simulation project. Many experimental frames can be formulated for the same system (both source system and model) and the same experimental frame may apply to many systems. Why would we want to define many frames for the same system? Or apply the same frame to many systems? For the same reason that we might have different objectives in modeling the same system, or have the same objective in modeling different systems. There are two equally valid views of an experimental frame. One, views a frame as a definition of the type of data elements that will go into the database. The second views a frame as a system that interacts with the system of interest to obtain the data of interest under specified conditions. In this view, the frame is characterized by its implementation as a measurement system or observer. In this implementation, a frame typically has three types of components (as shown in Exhibit 13a): *generator* that generates input segments to the system; *acceptor* that monitors an experiment to see the desired experimental conditions are met; and *transducer* that observes and analyzes the system output segments.

Exhibit 13b) illustrates a simple, but ubiquitous, pattern for experimental frames that measure typical job processing performance metrics, such as relate to round trip time and throughput. Illustrated in the web context, a generator produces service request messages at a given rate. The time that has elapsed between sending of a request and its return from a server is the round trip

time. A transducer notes the departures and arrivals of requests allowing it to compute the average round trip time and other related statistics, as well as the throughput and unsatisfied (or lost) requests. An acceptor notes whether performance achieves the developer's objectives, for example, whether the throughput exceeds the desired level and/or whether say 99% of the round trip times are below a given threshold.

Objectives for modeling relate to the role of the model in systems design, management or control. Experimental frames translate the objectives into more precise experimentation conditions for the source system or its models. A model is expected to be valid for the system in each such frame. Having stated our objectives, there is presumably a best level of resolution to answer the questions raised. It is usually the case that the more demanding the question, the greater the resolution needed to answer it. Thus, the choice of appropriate levels of abstraction also hinges on the objectives and their experimental frame counterparts.

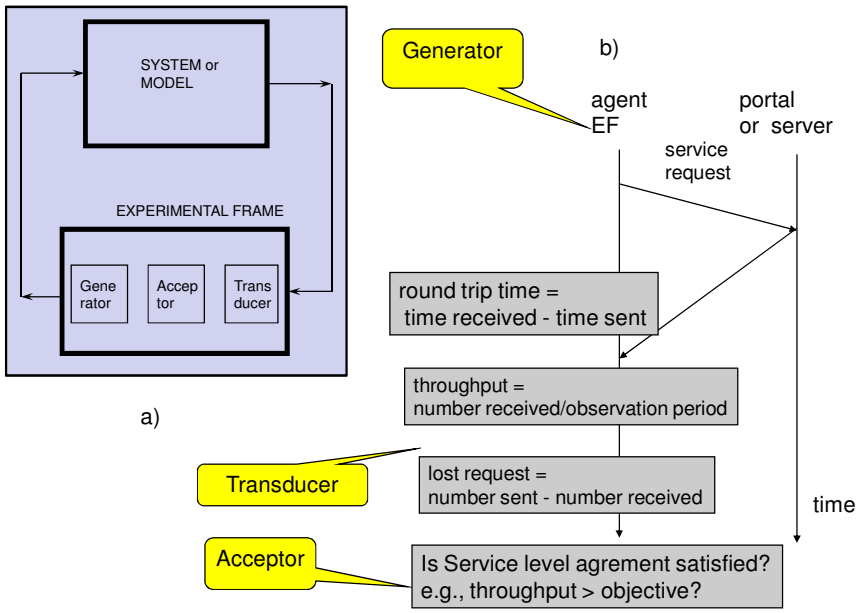

Exhibit 13: Experimental Frame and Components

Methods for transforming objectives into experimental frames have been discussed in the literature [ZEI00], [TRAO].. In the context of SoS engineering, modeling objectives support development and testing of SoSs. Here we need to formulate measures of the effectiveness (MOE) of a SoS in accomplishing its goal in order to allow us to rigorously evaluate the architectural and design alternatives. We call such measures, *outcome* measures. In order to compute such measures, the model must include certain variables; we'll call *output* variables, whose values are computed during execution runs of the model. The mapping of the output variables into outcome measures is performed by the transducer component of the experimental frame. Often there may be more than one layer of variables intervening between output variables and outcome measures. For example, in military simulations, measures of performance (MOP) are output variables that typically judge how well parts of a system are operating. Such measures enter as factors into MOEs.

7. EXPERIMENTAL FRAMES FOR SOS TEST AND EVALUATION

The DoD's concept of Jointness, in which assets of Army, Navy, Air Force, and other military services are brought together to execute a mission, offers a progenitor for numerous SoS examples. Joint Critical mission threads for a proposed SoS are intended to capture how the SoS's capabilities will be used in real world situations. The capability to measure effectiveness of joint missions requires the ability to execute such mission threads in operational environments, both live and virtual. Experimental Frame concepts offer an approach to enable simulated and real-time observation of participant system information exchanges and processing of acquired data to allow mission effectiveness to be assessed. Relevant MOPs include quality of shared situational awareness, quality and timeliness of information, and extent and effectiveness of collaboration. MOEs concern measures of the desired benefits such as increase in combat power, increase in decision-making capability, and increase in speed of command. Such metrics must be modeled with mathematical precision so as to be amenable to collection and computation with minimally intrusive test infrastructure to support rigorous, repeatable and consistent testing and evaluation (T&E). An example of a mission thread is the Theater Strategic Head Quarter Planning at the Combatant Command Level [NSB00].

As illustrated in Exhibit 14, this thread starts when the Combatant Command HQ receives a mission which is delegated to HQ Staff. The staff initiates the Mission Analysis Process which returns outputs and products to the Commander. Then the Joint planning group and the development teams develop courses of action and perform effects analysis, respectively. The thread ends with the issue of operational orders. The MOP of interest might include various ways of measuring extent and effectiveness of collaboration, while the MOE might be the increase in speed of command.

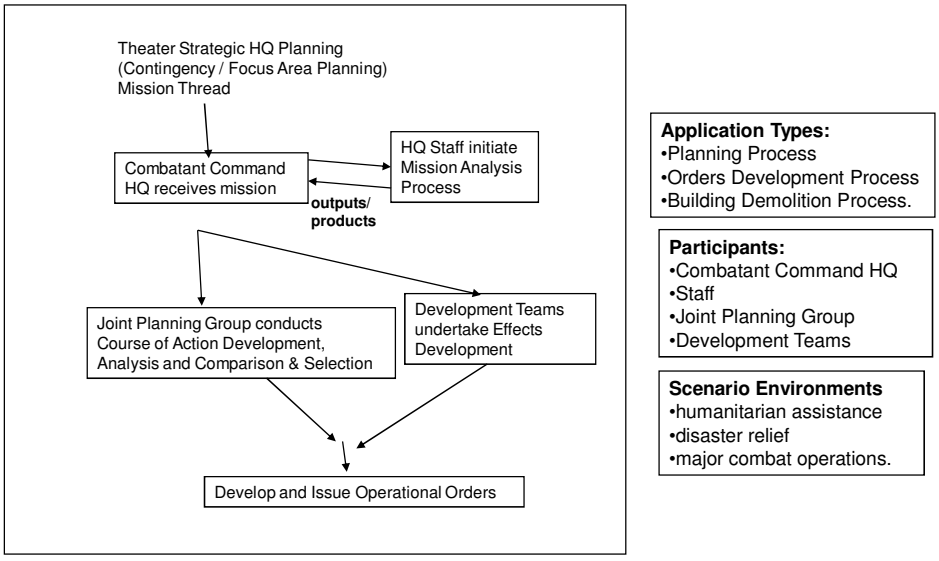

Exhibit 14: Illustrating a Joint Mission Thread and its dimensions for variation

These measures might be collected in assessing the value added of a collaboration support system in an operationally realistic setting. An informally presented mission thread can be regarded as a template for specifying a large family of instances. As illustrated in Exhibit 15, such instances can vary in several dimensions, including the objectives of interest (including the desired MOP and MOE), the type of application, the participants involved, and the operational environments in which testing will occur. Furthermore, mission threads can be nested, so that for example, Mission Analysis is a sub-thread that is executed within the Theater Planning thread. An instance of a collaboration mission thread can be modeled as a coupled model in DEVS (Recall Exhibit 1) in which the components are participants in the collaboration and couplings represent the possible information exchanges that can occur among them.

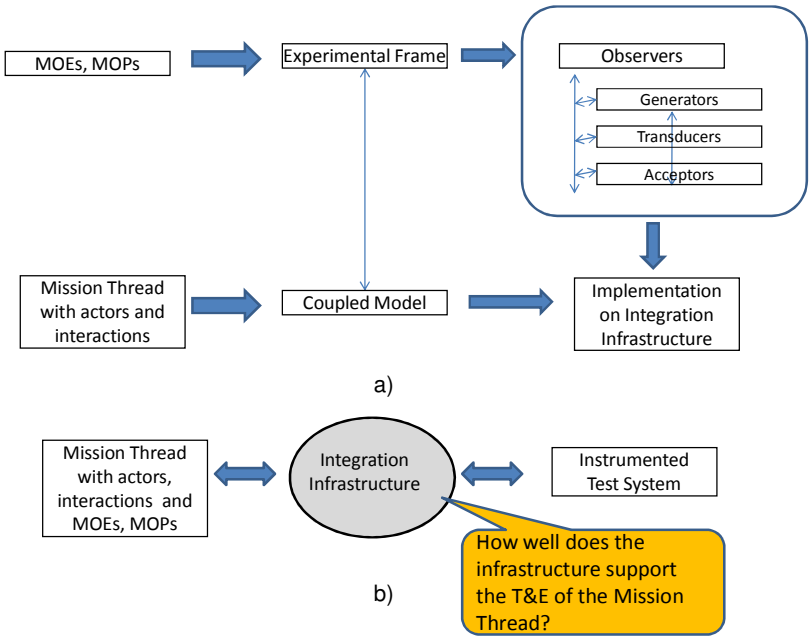

Exhibit 15: Mission thread implementation in an integration environment

This formulation offers an approach to capturing and implementing joint mission threads in the form of test model federations that can be deployed in a net-centric *integration infrastructure* such as the Global Information Grid/Service Oriented Architecture (GIG/SOA) [CHA05]. As illustrated in Exhibit 15a), such an infrastructure offers environment in which to deploy the participants in a mission thread together with network and web services that allow them to collaborate to achieve the mission objectives. As we shall see, formulation of a mission thread instance as a coupled model allows us to provide a rigorous method for realizing the thread within the integration infrastructure. At the same time, the MOEs and MOPs that have been formulated for assessing the outcome of the mission execution need to be translated into an appropriate experimental frame. In this case, the distributed nature of the execution will often require the frame to be distributed as well. The distributed frame will have such components as *observers* for the participant activities and message exchanges as well as the generators, transducers and acceptors previously discussed. Indeed, as in Exhibit 15b), the problem of assessing how well an integration infrastructure supports the collaboration requirements of a mission thread can be formulated as one of designing a *Instrumented Test System* where the latter is taken as an SoS and which therefore, can be addressed with M&S approach discussed here.

8. DEVS UNIFIED PROCESS AND ITS SERVICE ORIENTED IMPLEMENTATION

This section describes the refined bifurcated Model-Continuity process and how various elements like automated DEVS model generation, automated test-model generation (and net-centric simulation over SOA are put together in the process, resulting in DEVS Unified Process (DUNIP) [MIT07g]. The DEVS Unified Process (DUNIP) is built on the bifurcated Model-continuity based life-cycle methodology. The design of simulation-test framework occurs in parallel with the simulation-model of the system under design. The DUNIP process consists of the following elements:

1. Automated DEVS Model Generation from various requirement specification formats
2. Collaborative model development using DEVS Modeling Language (DEVSML)
3. Automated Generation of Test-suite from DEVS simulation model
4. Net-centric execution of model as well as test-suite over SOA

Considerable amount of effort has been spent in analyzing various forms of requirement specifications, viz, state-based, Natural Language based, Rule-based, BPMN/BPEL-based and DoDAF-based, and the automated processes which each one should employ to deliver DEVS hierarchical models and DEVS state machines [MIT07g]. Simulation execution today is more than just model execution on a single machine. With Grid applications and collaborative computing the norm in industry as well as in scientific community, a net-centric platform using XML as middleware results in an infrastructure that supports distributed collaboration and model reuse. The infrastructure provides for a platform-free specification language DEVS Modeling Language (DEVSML) [MIT07e] and its net-centric execution using Service-Oriented Architecture called DEVS/SOA [MIT07f]. Both the DEVSML and DEVS/SOA provide novel approaches to integrate, collaborate and remotely execute models on SOA. This infrastructure supports automated procedures is the area of test-case generation leading to test-models. Using XML as the system specifications in rule-based format, a tool known as Automated Test Case Generator (ATC-Gen) was developed which facilitated the automated development of test models[MAK06,ZEI05,MAK07].

The integration of DEVSML and DEVS/SOA is performed with the layout as shown below in Exhibit 16.

Various model specification formalisms are supported and mapped into DEVSML models including UML state charts [JLR07], a Exhibit driven state-based approach[MIT07h], Business Process Modeling Notation (BPMN) [BPM,BPEL] or DoDAF-based[MIT06a]. A translated DEVSML model is fed to the DEVSML client that coordinates with the DEVSML server farm. Once the client has DEVSJAVA models, a DEVSML server can be used to integrate the client's model with models that are available at other sites to get an enhanced integrated DEVSML file that can produce a coupled DEVSML model. The DEVS/SOA enabled server can use this integrated DEVSML file to deploy the component models to assigned DEVS web-server simulated engines. The result is a distributed simulation, or alternatively, a real-time distributed execution of the coupled model.

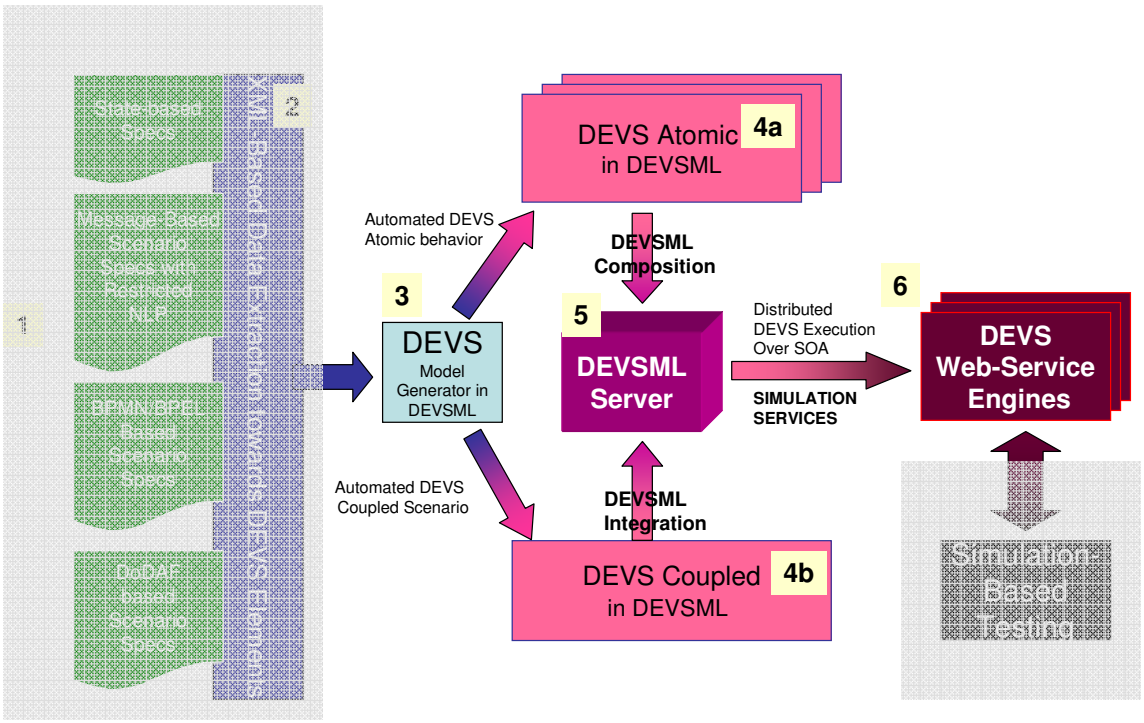

Exhibit 16: Net-centric collaboration and execution using DEVSML and DEVS/SOA

8.1 DEVSML Collaborative Development

DEVSML is a way of representing DEVS models in XML language. This DEVSML is built on JAVAML [BAD05], 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 transformable back'n forth to java and to DEVSML. It is an attempt to provide interoperability between various models and create dynamic scenarios. The key concept is shown in the Exhibit 16.

The layered architecture of the said capability is shown in Exhibit 17. At the top is the application layer that contains model in DEVS/JAVA or DEVSML. The second layer is the DEVSML layer itself that provides seamless integration, composition and dynamic scenario construction resulting in portable models in DEVSML that are complete in every respect. These DEVSML models can be ported to any remote location using the net-centric infrastructure and be executed at any remote location. Another major advantage of such capability is total simulator 'transparency'. The simulation engine is totally transparent to model execution over the net-centric infrastructure. The DEVSML model description files in XML contains meta-data information about its compliance with various simulation 'builds' or versions to provide true interoperability between various simulator engine implementations. This has been achieved for at least two independent simulation engines as they have an underlying DEVS protocol to adhere to. This has been made possible with the implementation of a single atomic DTD and a single coupled DTD that validates the DEVSML descriptions generated from these two implementations. Such run-time interoperability provides great advantage when models from different repositories are used to compose bigger coupled models using DEVSML seamless integration capabilities. More details about the implementation can be seen at [MIT07e]

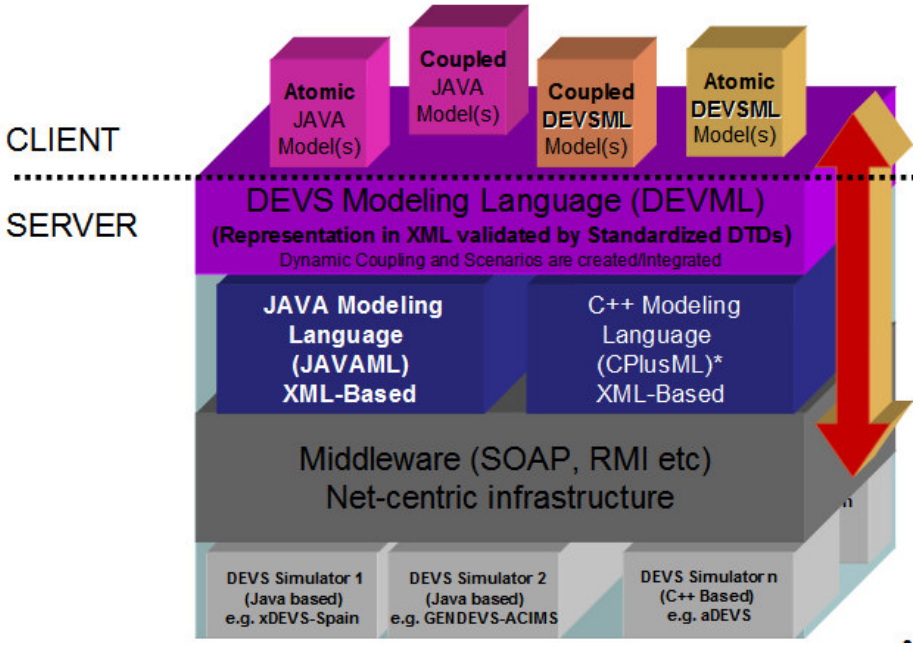

Exhibit 17: DEVS Transparency and Net-centric model interoperability using DEVSML. Client and Server categorization is done for DEVS/SOA implementation

8.2 DEVS/SOA: Net-centric Execution using Simulation Service

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

The complete setup requires one or more servers that are capable of running DEVS Simulation Service, as shown in Exhibit 18. The capability to run the simulation service is provided by the server side design of DEVS Simulation protocol supported by the latest DEVSJAVA Version 3.1[ACI06]. 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.

From multifarious modes of DEVS model generation, the next step is the simulation of these models. The DEVS/SOA client takes the DEVS models package and through the dedicated servers hosting simulation services, it performs the following operations:

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

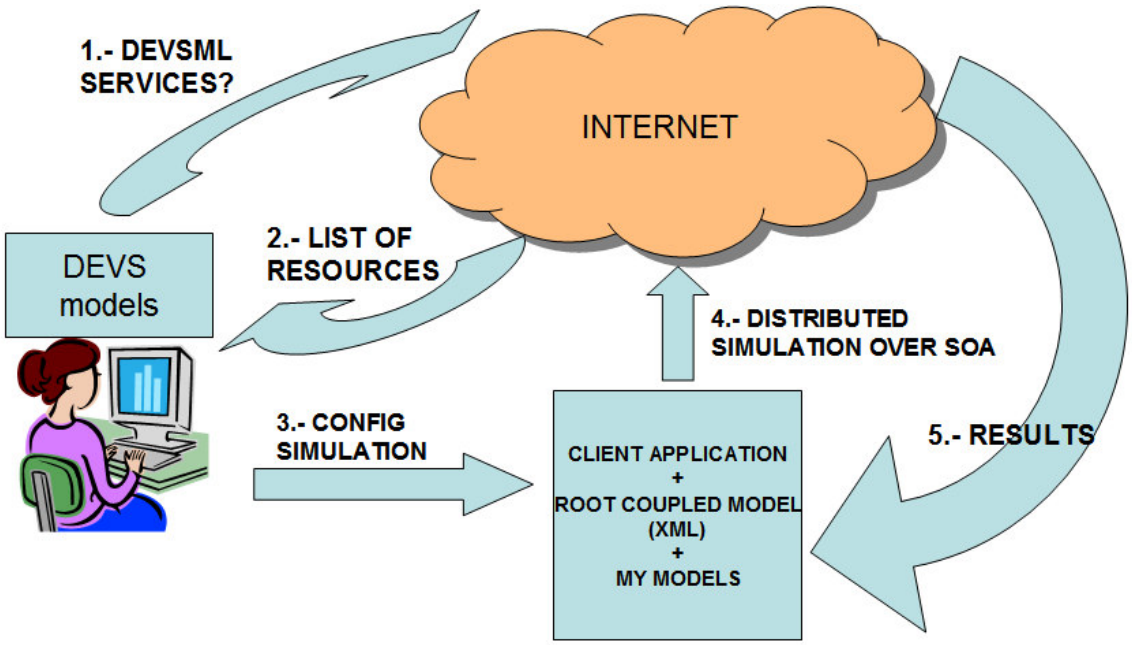

Exhibit 18: Execution of DEVS SOA-Based M&S

The main server selected (corresponding to the top-level coupled model) creates a coordinator that creates simulators in the server where the coordinator resides and/or over the other remote servers selected. The DEVS/SOA web service client as shown in Exhibit 19 below operates in the following sequential manner:

1. The user selects the DEVS package folder at his machine
2. The top-level coupled model is selected as shown in Exhibit 19
3. Various available servers are selected. Any number of available servers can be selected. Exhibit 20 shows how Servers are allocated on per-model basis. The user can specifically assign specific IP to specific models at the top-level coupled domain. The *localhost* (Exhibit 19) is chosen using debugging sessions.
4. The user then uploads the model by clicking the Upload button. The models are partitioned in a round-robin mechanism and distributed among various chosen servers
5. The user then compiles the models by clicking the Compile button at server's end
6. Finally, Simulate button is pressed to execute the simulation using Simulation service hosted by these services.
7. Once the simulation is over, the console output window displays the aggregated simulation logs from various servers at the client's end.

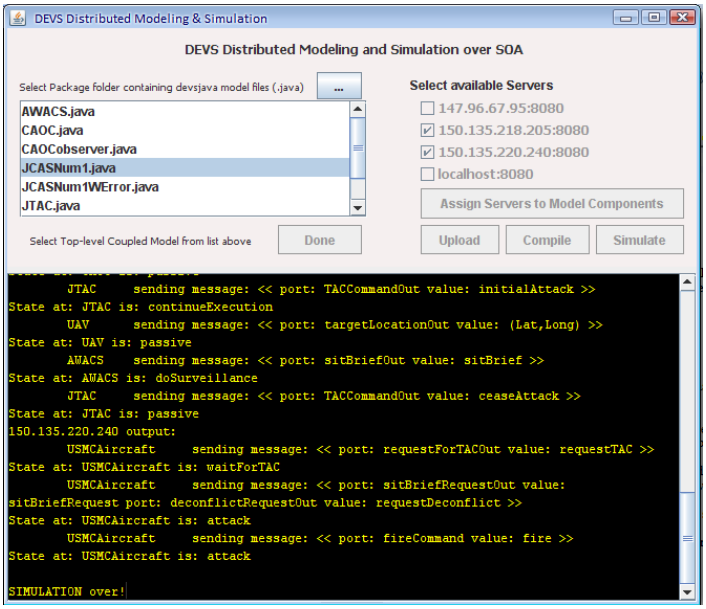

Exhibit 19: GUI snapshot of DEVS/SOA client hosting distributed simulation

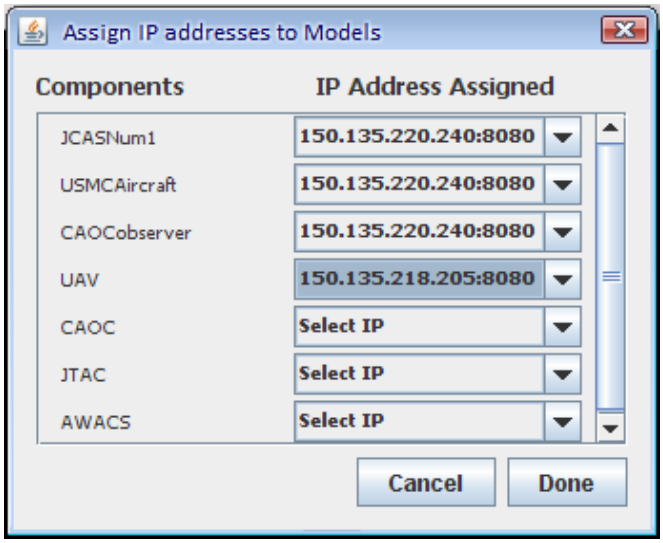

Exhibit 20: Server Assignment to Models

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

1. Specify the scenario
2. Develop the DEVS model
3. Develop the test-model from DEVS models
4. Run the model and test-model over SOA
5. Execute as a real-time simulation

6. Replace the model with actual web-service as intended in scenario.
7. Execute the test-models with real-world web services
8. Compare the results of steps 5 and 7.

One other section that requires some description is the multi-platform simulation capability as provided by DEVS/SOA framework. It consists of realizing distributed simulation among different DEVS platforms or simulator engines such as DEVSJAVA, DEVS-C++, etc on windows or linux platforms. In order to accomplish that, the simulation services will be developed that are focused on specific platforms, however, managed by a coordinator. In this manner, the whole model will be naturally partitioned according to their respective implementation platform and executing the native simulation service. This kind of interoperability where multi-platform simulations can be executed with our DEVSML integration facilities. DEVSML will be used to describe the whole hybrid model. At this level, the problem consists of message passing, which has been solved in this work by means of an adapter pattern in the design of the "message" class [MIT07f]. Exhibit 21 shows a first approximation. The platform specific simulator generates messages or events, but the simulation services will transform these platform-specific-messages (PSMsg) to our current platform-independent-message (PIMsg) architecture developed in DEVS/SOA.

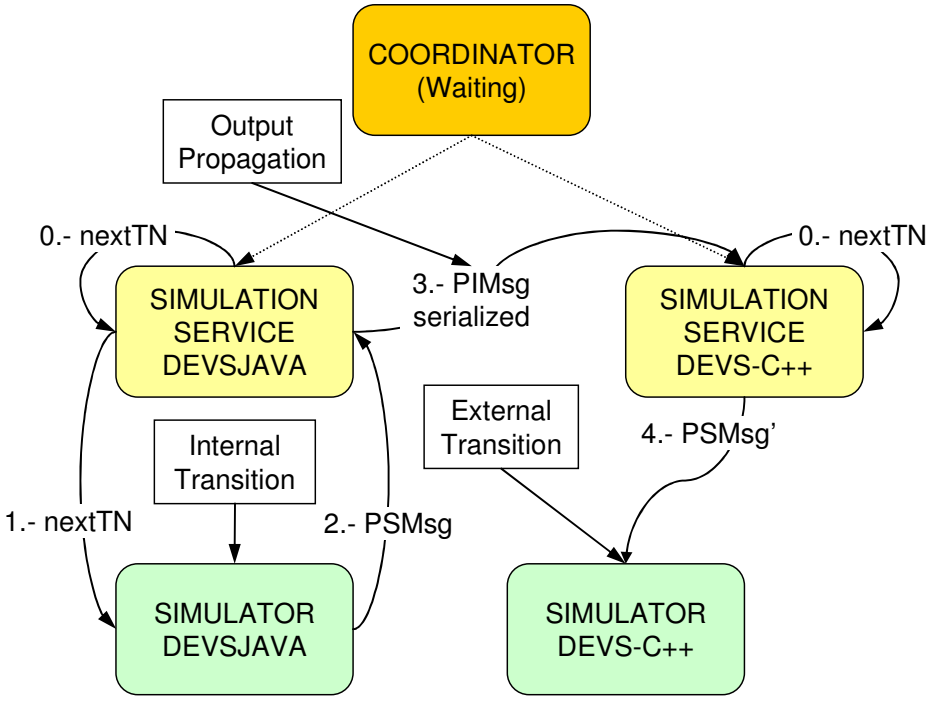

Exhibit 21: Interoperable DEVS Simulation protocol in DEVS/SOA

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

8.3 The Complete DEVS Unified Process

DUNIP can be summarized as the sequence of the following steps:

1. Develop the requirement specifications in one of the chosen formats such as BPMN, DoDAF, Natural Language Processing (NLP) based, UML based or simply DEVS-based for those who understand the DEVS formalism
2. Using the DEVS-based automated model generation process, generate the DEVS atomic and coupled models from the requirement specifications using XML
3. Validate the generated models using DEVS W3C atomic and coupled schemas to make them net-ready capable for collaborative development, if needed. This step is optional but must be executed if distributed model development is needed. The validated models which are Platform Independent Models (PIMs) in XML can participate in collaborative development using DEVSML.
4. From step 2, either the coupled model can be simulated using DEVS/SOA or a test-suite can be generated based on the DEVS models.
5. The simulation can be executed on an isolated machine or in distributed manner using SOA middleware if the focus is net-centric execution. The simulation can be executed in real-time as well as in logical time.
6. The test-suite generated from DEVS models can be executed in the same manner as laid out in Step 5.
7. The results from Step 5 and Step 6 can be compared for V&V process.

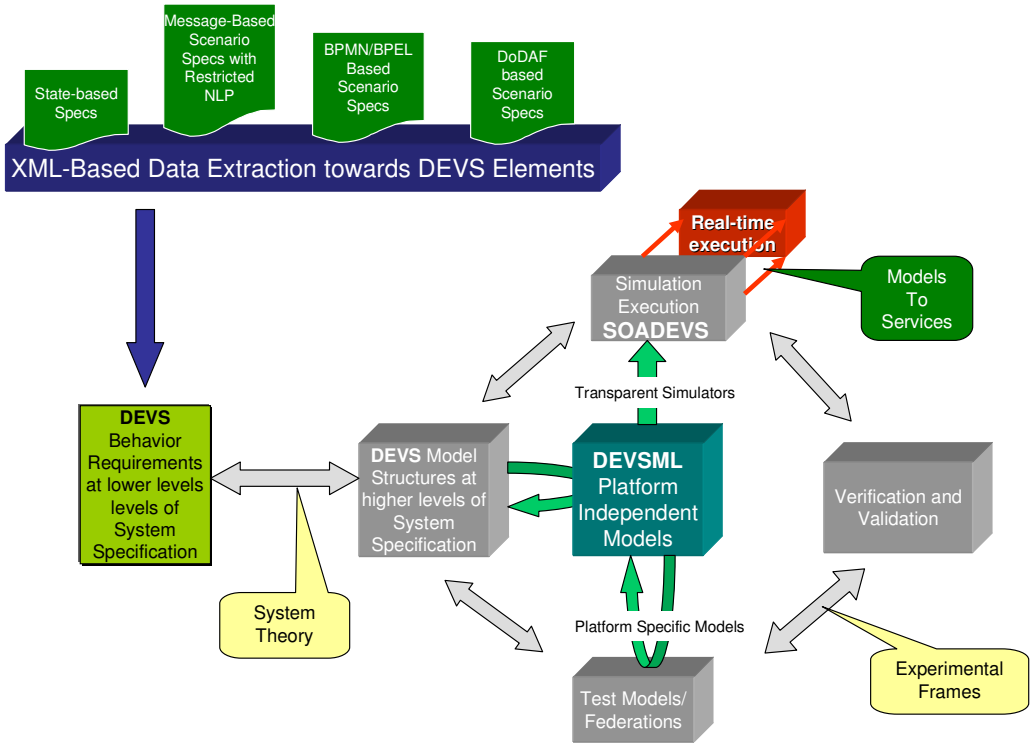

Exhibit 22: The Complete DEVS Unified Process

The basic Bifurcated Model Continuity-based Life-cycle process for systems engineering in Exhibit 10 in light of the developments in DEVS area is summarized in Exhibit 22 above. The grey boxes show the original process and the colored boxes show the extensions that were developed to make it a DEVS compliant process. A sample demo movie is available at [Dun07].

Many case studies came about as DUNIP was defined and developed. Many of the projects are currently active at Joint Interoperability Test Command (JITC) and others are at concept validation stage towards a deliverable end. Each of the projects either uses the complete DUNIP process or a subset of it. As we shall see on a case by case basis, DEVS emerge as a powerful M&S framework contributing to the complete systems software engineering process. With the proposed DEVS Based Bifurcated Model-continuity Life-cycle process, systems theory with its DEVS implementation can support the next generation net-centric application development and testing.

A recent Doctoral Thesis [MIT07g] provides a number of examples that illustrate how the DUNIP can be applied to import real-world simulation-based design applications. Here we review briefly the following case studies that were developed in more detail in [MIT07g]:

- Joint Close Air Support (JCAS) model
- DoDAF-based Activity scenario
- Link-16 Automated Test Case Generator (ATC-Gen project at JITC)
- Generic Network for Systems Capable of Planned Expansion (GenetScope project at JITC)

Each of the projects has been developed independently and ATC-Gen and GenetScope are team projects. All of the projects stand-alone and each applies DUNIP (Exhibit 22 in full or in-part. Exhibit 23 below provides an overview of the DUNIP elements used in each of the projects. All of the DUNIP elements have been applied at least once in one of the projects.

Project / DUNIP Elements		**JCAS model**	**DoDAF-based Activity Scenario**	**ATC-Gen Project**	**GenetScope Project**
Requirement Specification Formats		X			X
	State-based Specs	X			
	Message-based Specs with restricted NLP	X			
	BPMN/BPEL based Specs	X			
	DoDAF-Based Scenario Specs		X		X
XML-based Data Extraction		X	X	X	
DEVS Model Structure at lower levels of Specification		X	X	X	
DEVS model structure at higher levels of System specification			X		X
DEVSML Platform Independent		X			

Models				
Test Model Development	X		X	
Verification and Validation using Experimental Frames		X	X	X
DEVS/SOA net-centric Simulation	X			

Exhibit 23: Overview of DUNIP application in available case-studies

The JCAS system requirements come in many formats and served as a base example to test many of the DUNIP processes for requirements-to-DEVS transformation. JCAS requirements were specified using the state-based approach, BPEL-based approach and restricted natural language approach. The JCAS case study describes how each of the three approaches led to executable DEVS models with identical simulation results. Finally, the simplest executable model (that specified by the state-based approach) was executed over a net-centric platform using DEVSML and DEVS/SOA architecture.

The DODAF-based Activity scenario was specified using the UML-based Activity diagrams. It illustrates the process needed to transform various DoDAF documents into DEVS requirement specifications. New Operational View documents OV-8 and OV-9 were proposed [Mit06a] to facilitate the transformation of DoDAF requirements into a form that could be supported by DEVS-based modeling and simulation. The population of these new documents was described as well as how DEVS models could be generated from them.

The ATC-Gen project at JITC is the project dealing with automated Link-16 testing environment and the design of ATC-Gen tool. A detailed discussion and complete example are presented in [MAK06].

The GenetScope project at JITC is another project that employs the complete DEVS software engineering process. Using automated XML data mining, a ten year-old legacy model written in the C language was transformed to an object-oriented DEVS model with enhanced Model View Simulation and Control paradigm [Mit06b]. The design elements of GENETSCOPE tool were discussed and as was its relationship with the overarching DoDAF framework [Mit06b].

Although the DUNIP was applied in part to each of the above projects, presently, there is no live case study that implements all the aspects of DUNIP elements. Recall that DUNIP was researched and developed in the context of the above active projects.

To summarize, with the DEVS Unified Process we have the capability to:

- Transform various forms of requirement specifications to DEVS models in an automated manner.
- Transform any DEVS model to a Platform Independent Model (PIM) by using DEVSML for model and library reuse and sharing thus supporting collaborative development
- Simulate any valid DEVSML using the DEVS/SOA architecture thus exploiting the abstract DEVS simulator paradigm to achieve interoperability of DEVS model execution (for models implemented in disparate languages e.g. Java and C++)
- Transform any DEVSML model to a Service component in SOA and any coupled model into a deployable collaboration of such Service components.

9. APPLICATION: SYSTEM OF SYSTEMS SIMULATION FOR HETEROGENEOUS MOBILE SENSOR NETWORKS

In real world systems, the system of systems concept is addressed in a higher level where the systems send and receive data from other systems in the SoS and make a decision that leads the SoS to its global goals. Let's take the military surveillance example where different units of the army collect data through their sensors trying to locate a threat or determine the identity of a target. In this type of situations, army command center receives data from these heterogeneous sensor systems such as AWACS, ground RADARS, submarines, and so on. In general, these systems are different in hardware and/or software. This will create a huge barrier in data aggregation and data fusion using the data received from these systems because they would not be able to interact successfully without hardware and/or software compatibility. In addition, the data coming from these systems are not unified which also adds to the barrier in data aggregation.

One solution to the problem is to modify the communication medium among the SoS components. Two possible ways of accomplishing this task are [SAH07A]:

- *Create a software model of each system*. In this approach, each component in the SoS talks to a software module embedded in itself. The software module collects data from the system and through the software model generates outputs and sends to the other SoS components. If these software modules are written with a common architecture and a common language, then the SoS components can communicate effectively regardless of their internal hardware and/or software architectures.
- *Create a common language to describe data*. In this approach, each system can express its data in this common language so that other SoS components can parse the data successfully.

The overhead that needs to be generated to have software models of each system on a SoS is enormous and must be redone for new member of the SoS. In addition, this requires the complete knowledge of the state-space model of each SoS components, which is often not possible. Thus, data driven approach would have better success on integrating new members to the SoS and also applying the concept to other SoS application domains.

In this work, we present SoS architecture based on Extensible Markup Language (XML) in order to wrap data coming from different sources in a common way. The XML language can be used to describe each component of the SoS and their data in a unifying way. If XML based data representation architecture is used in a SoS, only requirement for the SoS components to understand/parse XML file received from the components of the SoS.

In XML, data can be represented in addition to the properties of the data such as source name, data type, importance of the data, and so on. Thus, it does not only represent data but also gives useful information which can be used in the SoS to take better actions and to understand the situation better. The XML language has a hierarchical structure where an environment can be described with a standard and without a huge overhead. Each entity can be defined by the user in the XML in terms of its visualization and functionality. For example, a hierarchical XML architecture like in Listing 1 can be designed for a SoS so that it can be used in the components of the SoS and also be applied to other SoS domains easily. In Listing 1, the first line defines the name of the file that describes the functionality of the user-defined keywords used to define the SoS architecture. This file is mainly used for visualization purposes so that any of the SoS components can display the current data or the current status of the SoS to a human/expert to make sure the proper decision is taken.

```
<!--Created 11/8/2006   Author @ Ferat Sahin-->
<?xml-stylesheet type="text/css" href="genericxml.css"?>

<systemofsystem>
        <id> Id of the System of systems </id>
        <name> The name of the System of System</name>
        <system>
                <id>Id of the first system</id>
                <name> The name of the first system </name>
                <description>  The description of the first system </description>
                <dataset>
                        <Output>
                                <id>Id of the first output</id>
                                <data>Data of the first output</data>
                        </Output>
                        <Output>
                                <id>Id of the second output</id>
                                <data>Data of the second output</data>
                        </Output>
                </dataset>
                <subsystem>
                        <id>Id of the subsystem of the first System</id>
                        <name>The name of the subsystem</name>
                        <description>This is a subsystem of the system in a SoS</description>
                        <dataset>
                                <Output>
                                        <data> Data of the subsystem </data>
                                </Output>
                        </dataset>
                </subsystem>
        </system>
</systemofsystem>
```

Exhibit 24: An XML based System of systems architecture

In Exhibit 24, the first keyword of the XML architecture is “systemofsystem” representing a SoS. Everything described after this keyword belongs to this SoS based on the XML architecture. The following keywords, “id” and “name”, are used to describe the system of systems.

Then, the keyword “system” is used to declare and describe the first system of the SoS. In addition to “id” and “name”, two more keywords, “description” and “dataset”, are used to describe the properties of the system and to represent the data coming out of this system. Data sources are denoted by “Output” keyword and data is provided with the keyword “data”. After representing data from two sources, a subsystem is described by the keyword “subsystem”. The subsystem and its data are presented in a similar manner. Other subsystems can be described in this subsystem or in parallel to this subsystem as well as additional systems can be described in the system of systems.

Next, we will present a simulation case-study to demonstrate SoS simulation framework for threat detection in a heterogeneous mobile sensor networks. In this scenario, there is a haptic controlled base robot (Exhibit 25a), a swarm robot, and two sensor modules, shown in Exhibit 25b. Before these systems are put in the field to do threat detection, the system of system

concepts should be simulated using suitable simulation tools to test the performance of these SoS concepts.

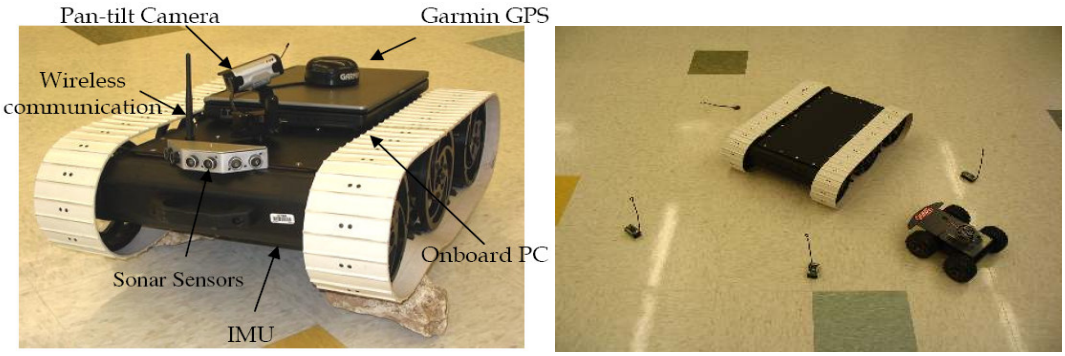

Exhibit 25: (a) Base robot with haptic control,
(b) Components of multi-sensor data aggregation and fusion system of systems [SAH07b].

In addition, a simulation environment can also assist us to evaluate our robust data aggregation, fusion, and decision-making techniques. The next section explores a possible simulation environment that can accommodate asynchronous system interactions and data exchange using XML.

9.1 SoS Simulation of Threat Detection (Data Aggregation)

Multi-agent robotic systems can also be successfully demonstrated using the DEVS formalism and DEVSJAVA software [SAR01]. A multi-agent simulation framework, Virtual Laboratory (V-Lab®) was developed for such multi-agent simulation with DEVS modeling framework [SRI04, SRI03].

Based on the tiered system of systems architecture [SRI06, SRI07], we have simulated a data aggregation scenario where there are two robot (a Base Robot, a Swarm robot), two sensors, and a threat [SAH07b]. When the sensors detect the threat, they notify the Base Robot about the presence of a threat. Upon receiving such notification, the Base Robot notifies the Swarm Robot the location of the threat based on the information sent by the sensors.

An XML based SoS message architecture is implemented in DEVSJAVA software [SAR00]. In this XML based message architecture [SAH07a, SAH07b], each system has an XML-like message consisting of their name and a data vector. The name of each system represents an XML tag. The data vectors are used to hold the data of the systems. The length of the vectors in each system can be different based on the amount of data each system contains. For instance, the XML message of the sensors has the sensor coordinates and the threat level. The threat level is set when a threat gets in the coverage area of the sensors. On the other hand, the Base Robot's XML message has its coordinates, the threat level, and the coordinates of the sensor who is reporting a threat. Thus, the data vector length of Base Robot XML message has five elements whereas the data vector of an XML message of a sensor has three elements. Exhibit 26 presents the names and the length of the vector data of each system in the system of systems.

The data vectors are made of "double" variables in order to keep track of the positions accurately. The "Threat" element in the data vector is a flag representing threat (1.0) or no threat (0.0). The elements "Xt" and "Yt" are used the destination coordinates in the XML messages of the Base Robot and Swarm Robot. These represent the coordinates of the sensor who reported a

threat. The threat is named as "Fire" for obvious reasons and it only has two elements in its data vector for its coordinates.

System	Name	Vector Data Length
Base Robot	"Base Robot"	5 (X, Y, Threat, Xt, Yt)
Swarm Robot	"Swarm Robot"	5 (X, Y, Threat, Xt, Yt)
Sensor	"Sensor 1"	3 (X, Y, Threat)
Sensor	"Sensor 2"	3 (X, Y, Threat)
Threat	"Fire"	2 (X, Y)

Exhibit 26: XML message components for the systems in the SoS.

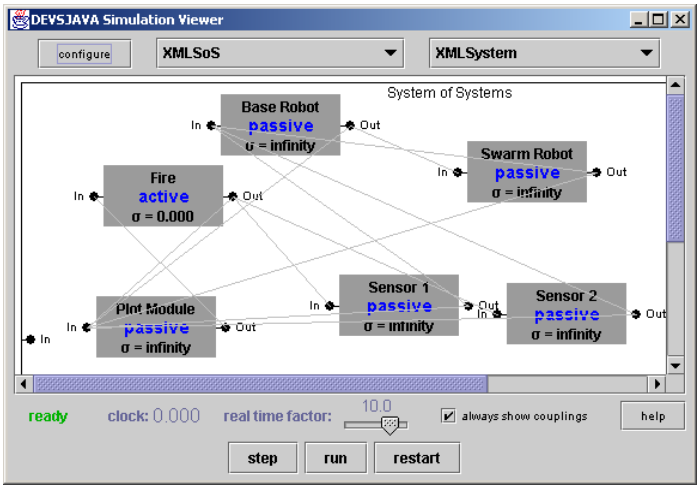

Exhibit 27: The DEVSJAVA atomic and coupled modules for XML base SoS simulation [SAH07a].

In order to generate these XML-like messages, a data structure, called "XmlEntity", is created based on the "entity" data structure in DEVSJAVA environment. This data structure is used to wrap/represent the data of each system. The structures/behaviors of the systems in the SoS are created/simulated by DEVSJAVA atomic or coupled models. Exhibit 27 is a screen shot of the DEVS model of the system of systems described above.

Exhibit 28 shows a DEVS simulation step with the XML messages sent among the systems in the system of systems. As can be seen in Exhibit 27, each system sends and XML-like messages which consist of the name of the system and the data vector related to its characteristics.

Finally, Exhibit 29 shows the simulation environment created by the "Plot Module" atomic model in DEVS. In the environment, the two sensors are located next to each other representing a border. The "Threat" ("Fire"), read dot, is moving in the area. When the threat is in one of the sensor's coverage area, the sensor signals the Base Robot. Then, Base Robot signals the Swarm Robot so that it can go and check whether the threat is real or not. The behavior of the system of systems components can also be seen in Exhibit 28 as the movements of the "Threat" and the "Swarm Robot" are captured. The green dot represents the Swarm Robot. The Base Robot is not shown since it does not move in the field. When the Threat enters into the coverage area of a sensor, that sensor area filled with read color to show the threat level in the coverage area. The sensor then reports the threat to the Base Robot. Then, the Base Robot issues a command to a Swarm Robot closest to the sensor area. Finally, the Swarm Robot, green dot, moves into the sensor coverage area based on the coordinates sent by the Base Robot to check whether the threat is real. When the Swarm Robot reaches the sensor coverage area it determines the threat level

with its own sensor and report it the Base Robot. If the threat is not real, the Swarm Robot moves away from the sensor's coverage area and waits another command from the base robot.

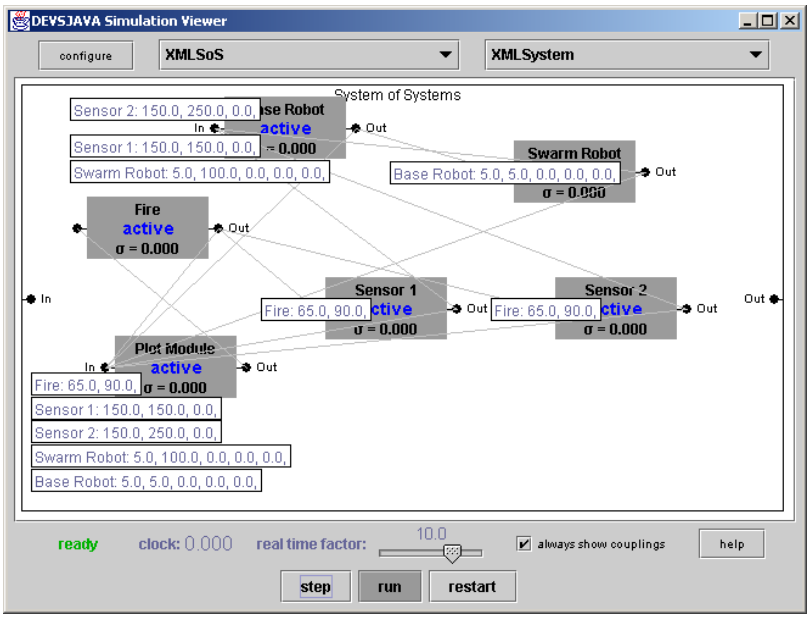

Exhibit 28: The DEVSJAVA simulation with XML based messages shown at the destination [SAH07b]

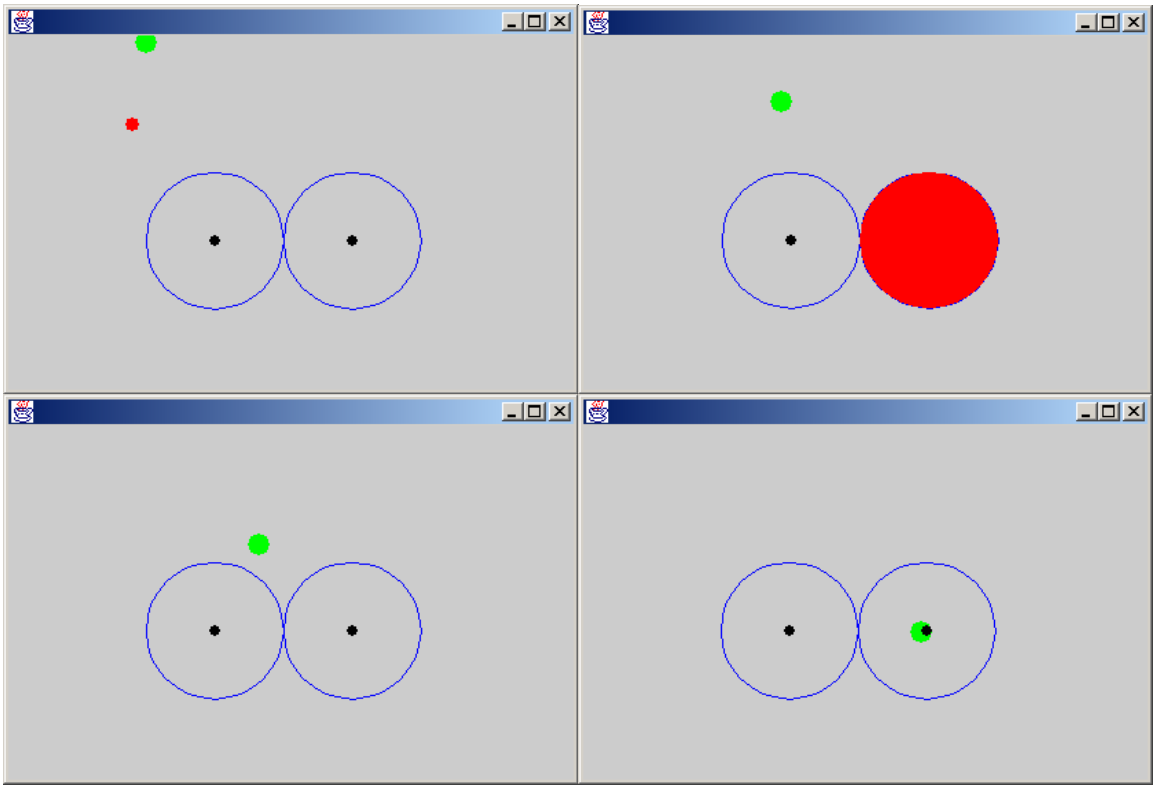

Exhibit 29: The progress of the DEVSJAV simulation on data aggregation

9.2 Concluding Remarks

In this section, we have presented a framework to simulate a system of systems and discussed a simulation case-study (threat detection) with multiple systems working collaboratively as a SoS. While DEVS formalism helps to represent the structure of a SoS, the XML provides a way to represent the data generated by each system. Together, DEVS formalism and XML form a powerful tool for simulating any given SoS architecture. To the best of our knowledge, there has been very little research directed towards the development of a *generic framework* for architectural representation and simulation of a SoS. Currently, we are working on extending the XML data representation in DEVS and making it more generic and dynamic so that when a new system is added to a SoS, it will automatically generate its XML message in order to send its data to other components of the SoS.

10. APPLICATION: AGENT-IMPLEMENTED TEST INSTRUMENTATION SYSTEM

The Test Instrumentation System should provide a minimally intrusive test capability to support rigorous, on-going, repeatable and consistent testing and evaluation (T&E). Requirements for such a test implementation system include ability to

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

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

Before proceeding we present an example that provides an exemplar of the design approach to follow.

10.1 Example: Collaboration Session Timing Instrumentation

The experimental frame coupled to the activity model through observers is shown in Exhibit 30. In the activity model, users can interact with each other through a collaboration service and also independently interact with a portal for other services (such as searching a database,

checking accounting data, etc.) Information about user activity is collected by observers, shown in pink, and individually coupled with users (black arrows).

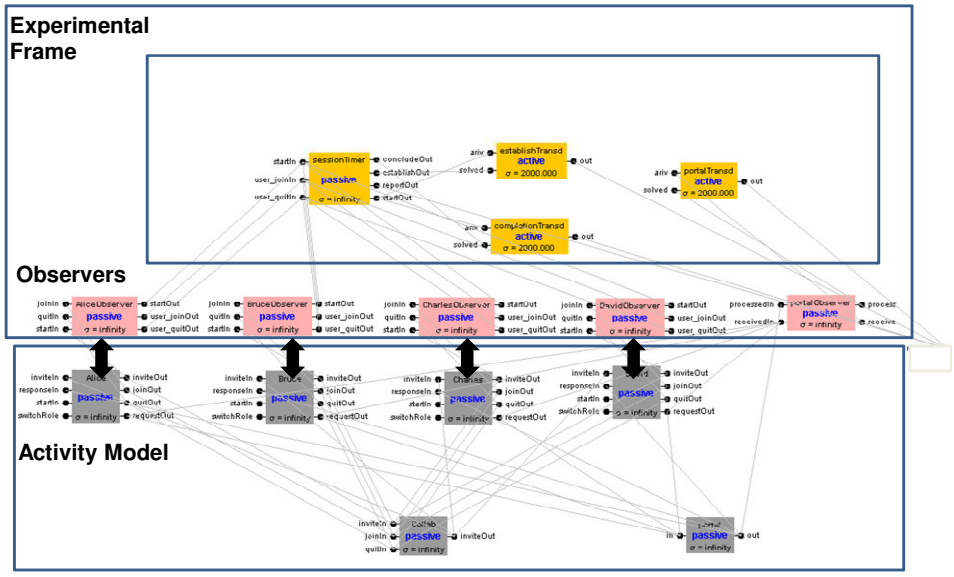

Exhibit 30: DEVSJAVA model of a collaboration

These observers are implemented as DEVS agents that are deployed in one-one fashion to user work sites. In the current example, these agents appears as applet clients with GUIs that enable users to generate status events by clicking a check box to indicate having just joined a collaboration session or having just quit one. Note that this information is obtained independently of the collaboration tool and does not rely on the latter's own status tracking. Such independent information, which might otherwise be obtained by human observers taking notes, is necessary to independently validate the collaboration tool's own logging and statistics computations. Status events generated by observers are sent to frame components, shown in yellow, for processing into performance statistics concerning the success rate in establishing collaboration sessions, the time required to establish such sessions and their duration. A somewhat different observer, that for portal services, collects user service requests sent to, and received from, the portal. This observer passes on such request messages to a frame component that computes throughput and response time statistics for portal services.

10.2 Distributed Test Federations

A DEVS distributed federation is a DEVS coupled model whose components reside on different network nodes and whose coupling is implemented through middleware connectivity characteristic of the environment, e.g., SOAP for GIG/SOA. The federation models are executed by DEVS simulator nodes that provide the time and data exchange coordination as specified in the DEVS abstract simulator protocol.

As discussed earlier, in the general concept of experimental frame (EF), the generator sends inputs to the SoS under test (SUT), the transducer collects SUT outputs and develops statistical summaries, and the acceptor monitors SUT observables making decisions about continuation or termination of the experiment [ZEI05]. Since the SoS is composed of system components, the EF is distributed among SoS components, as illustrated in Exhibit 31 a)**.** Each component may be coupled to an EF consisting of some subset of generator, acceptor, and transducer components. As mentioned, in addition an observer couples the EF to the component using an interface provided by the integration infrastructure. We refer to the DEVS model that consists of the observer and EF as a *test agent*.

Net-centric Service Oriented Architecture (SOA) provides a currently relevant technologically feasible realization of the concept. As discussed earlier, the DEVS/SOA infrastructure enables DEVS models, and test agents in particular, to be deployed to the network nodes of interest. As illustrated in Exhibit 31b), in this incarnation, the network inputs sent by EF generators are SOAP messages sent to other EFs as destinations; transducers record the arrival of messages and extract the data in their fields, while acceptors decide on whether the gathered data indicates continuation or termination is in order [MIT07].

Since EFs are implemented as DEVS models, distributed EFs are implemented as DEVS models, or agents as we have called them, residing on network nodes. Such a federation, illustrated in Exhibit 32, consists of DEVS simulators executing on web servers on the nodes exchanging messages and obeying time relationships under the rules contained within their hosted DEVS models.

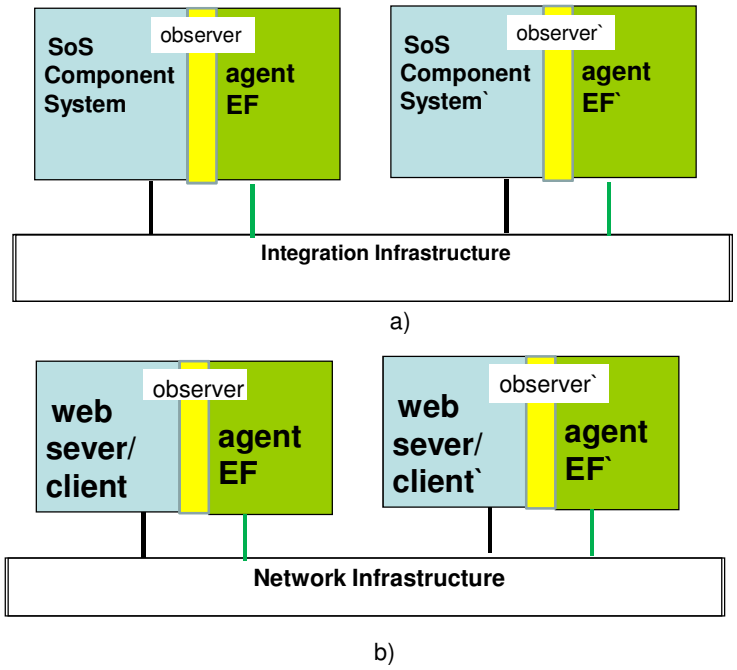

Exhibit 31: Deploying Experimental Frame Agents and Observers

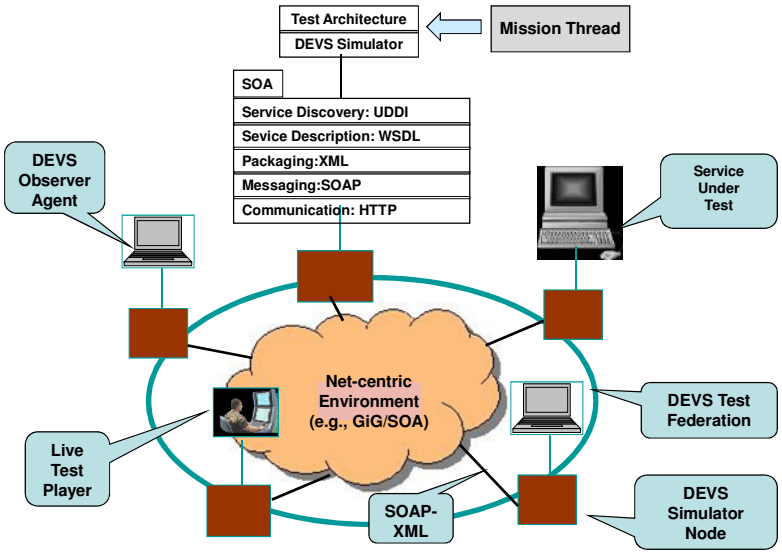

Exhibit 32: DEVS Test Federation in GIG/SOA Environment

10.3 Distributed Multi-level Test Federations

The linguistic levels of interoperability discussed earlier provide a basis for further structuring the test instrumentation system. In the following sections, we discuss the implementation of test federations that simultaneously operate at the syntactic, semantic, and pragmatic levels (Exhibit 33).

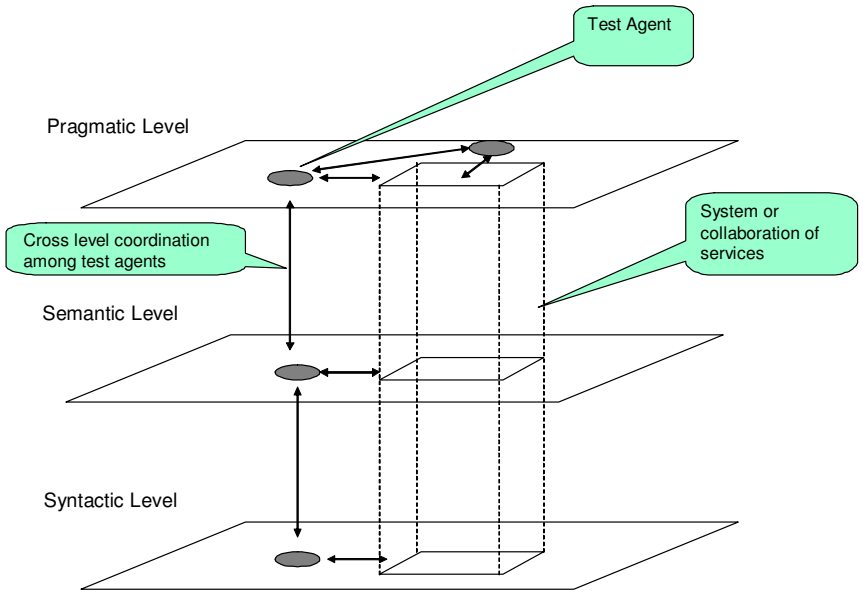

Exhibit 33: Simultaneous testing at multiple levels

10.3.1 Syntactic Level – Network Health Monitoring

From the syntactic perspective, testing involves assessing whether the infrastructure can support the speed and accuracy needed for higher level exchange of information carried by multimedia data types, individually and in combination. We now consider this as a requirement to continually assess whether the network is sufficiently "healthy" to support the ongoing collaboration. Exhibit 34 illustrates the architecture that is implied by the use of subordinate probes. Nodal generator agents activate probes to meet the health monitoring Quality of Service (QOS) thresholds determined from information supplied by the higher layer test agents, viz., the objectives of the higher layer tests.

Probes return statistics and alarm information to the transducers/acceptors at the DEVS health layer which in turn may recommend termination of the experiment at the test layer when QOS thresholds are violated.

In an EF for real-time evaluation of network health, the SUT is the network infrastructure (OSI layers 1-5) that supports higher session and application layers. QOS measures are at the levels required for meaningful testing at the higher layers to gather transit time and other statistics, providing quality of service measurements,

For messages expressed in XML and carried by SOAP middleware such messages are directly generated by the DEVS generators and consumed by the DEVS transducers/acceptors. Such messages experience the network latencies and congestion conditions experienced by messages exchanged by the higher level web servers/clients. Under certain QOS conditions however, video streamed and other data typed packets may experience different conditions than the SOAP-borne messages. For these we need to execute lower layer monitoring under the control of the nodal EFs.

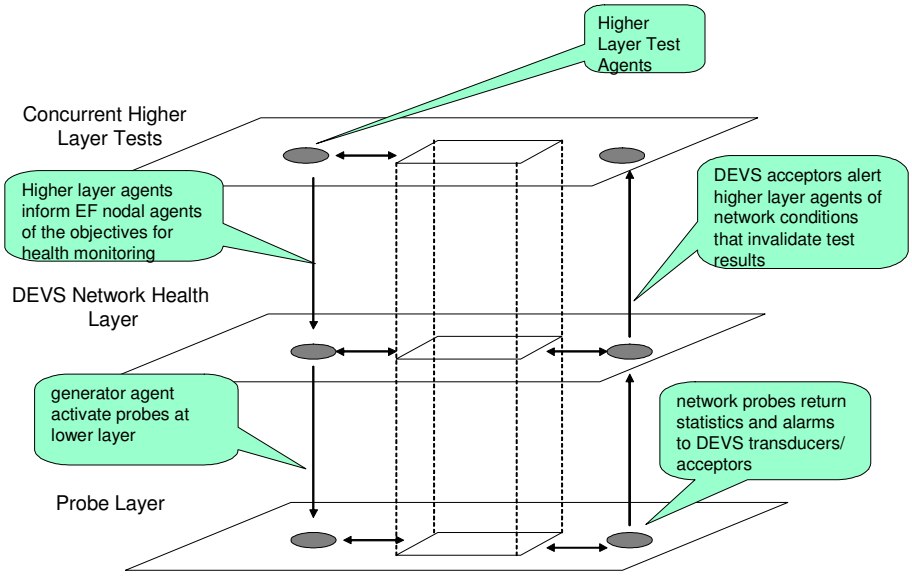

Exhibit 34: Multi-layer testing with Network Health Monitoring

The collection of agent EFs has the objective of assessing the health of the network relative to the QOS that it is providing for the concurrent higher level tests. Thus such a distributed EF is

informed by the nature of the concurrent test for which it monitoring network health. For example, if a higher level test involves exchanges of a limited subset of media data types (e.g., text and audio), then the lower layer distributed EF need only monitor the subset of types.

10.3.2 Semantic Level – Information Exchange in Collaborations

Mission threads consist of sequences of discrete information exchanges. A collaboration service supports such exchanges by enabling collaborators to employ a variety of media, such as text, audio, and video, in various combinations. For example, a drawing accompanied by a voice explanation involves both graphical and audio media data. Further, the service supports establishing producer/consumer relationships. For example, the graphical/audio combination might be directed to one or more participants interested in that particular item. From a multilevel perspective, testing of such exchanges involves pragmatic, semantic, and syntactic aspects. From the pragmatic point-of-view, the ultimate worth of an exchange is how well it contributes to the successful and timely completion of a mission thread. From the semantic perspective, the measures of performance involve the speed and accuracy with which an information item, such as a graphical/audio combination, is sent from producer to consumer. Accuracy may be measured by comparing the received item to the sent item using appropriate metrics. For example, is the received graphic/audio combination within an acceptable "distance" from the transmitted combination, where distance might be measured by pixel matching in the case of graphics and frequency matching in the case of audio. To automate this kind of comparison, metrics must be chosen that are both discriminative and quick to compute. Further, if translation is involved, the "meaning" of the item must be preserved as discussed above. Also, the delay involved in sending an item from sender to receiver, must be within limits set by human psychology and physiology. Such limits are more stringent where exchanges are contingent on immediately prior ones as in a conversation. Instrumentation of such tests is similar to that at the syntactic level to be discussed next, with the understanding that the complexity of testing for accuracy and speed is of a higher order at the semantic level.

10.3.3 Pragmatic Level – Mission Thread Testing

A test federation observes an orchestration of web-services to verify the message flow among participants adheres to information exchange requirements. A mission thread is a series of activities executed by operational nodes and employing the information processing functions of web-services. Test agents watch messages sent and received by the services that host the participating operational nodes. Depending on the mode of testing, the test architecture may, or may not, have knowledge of the driving mission thread under test. If a mission thread is being executed and thread knowledge is available, testing can do a lot more than if it does not.

With knowledge of the thread being executed, DEVS test agents can be aware of the current activity of the operational nodes it is observing. This enables an agent to focus more efficiently on a smaller set of messages that are likely to provide test opportunities.

10.3.4 Measuring Success in Mission Thread Executions

The ultimate test of effectiveness of an integration infrastructure is its ability to support successful outcomes of mission thread executions. To measure such effectiveness, the test instrumentation system must be informed about the events and messages to expect during an

execution, including those that provide evidence of success or failure, and must be able to detect and track these events and messages throughout the execution.

10.3.5 Measuring "the right information at the right place at the right time"

It is often said that the success of a mission depends on the right information arriving to the right place at the right time. Thus an obvious measure of performance is the ability of an integration infrastructure to measure the extent to which the right information is delivered to the right place at the right time. This in turn places requirements on the test instrumentation system that it be able to gather the information needed to make such judgments. Much more than the simple ability to determine mission success for failure, such a capability would provide diagnostic capability to determine whether the final outcome was due to failure in information flow, and if so, just what information did not get received at the right time by which consumer.

Exhibit 35 graphically illustrates a formulation of the issue at hand. Given that a consumer is engaged in a particular activity during some period, there is a time window during which it expects, and can exploit in its processing, some piece of information denoted by X. As in Exhibit 35a), if X arrives before this window opens up, it may too early and cannot be used when needed; likewise, if X arrives after the window has closed, it is too late for the information to be effectively used. Further, as in Exhibit 35b), if the wrong information, say ~X, arrives during the window of opportunity, it may be ignored by the consumer at best, or cause a back up of messages that clog the system at worst. As in Exhibit 35c), to make such determinations, a test agent has to know the current activity phase of the consumer, and have been informed of the expected information X and its time window of useability.

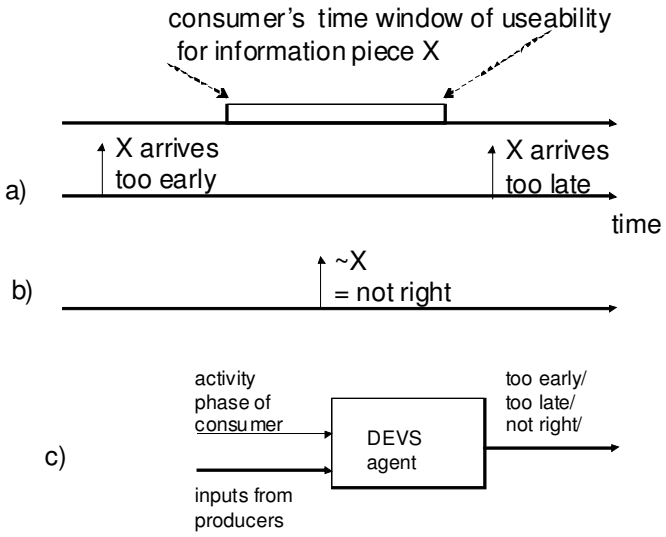

Exhibit 35: Instrumentation for "the right information at the right place at the right time"

Implementing such test capabilities requires not only the right test instrumentation and protocol but also the necessary back up analysis capabilities to provide the information items needed.

10.4 Analysis Capabilities

A mathematical model, such as the DEVS coupled model, allows the use of analysis tools to derive useful information to support testing. As illustrated in Exhibit 36, given a DEVS model of a mission thread, with upper and lower bounds on the times for individual activities, it is possible to derive time windows for the occurrence of events and arrival of messages. In the case of analysis, even more so than for simulation, it is important to down-select to the events and messages of particular interest, as derived from the MOPs and MOEs in the experimental frame. This is so because, unless appropriately constrained, analysis is typically intractable since it suffers from "global state explosion."

Algorithms are being developed that, under suitable restrictions of the class of models, can derive the time window specifications needed to inform the DEVS agents monitoring the related mission thread actors [MIT07g,ZEI05].

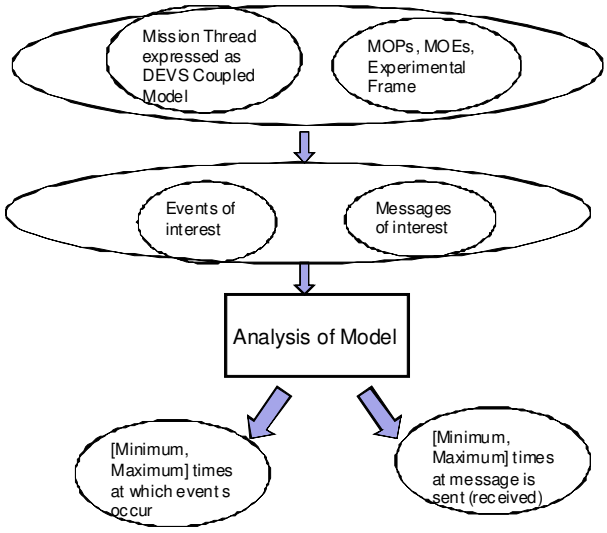

Exhibit 36: Analysis of Mission Thread models to derive test-supporting temporal information

10.5 Verification/Validation of the Test Instrumentation System

How do we validate/verify the Test Instrumentation System (TIS) itself? Exhibit 37 presents the development of such a system as itself an application of the DEVS Unified Process. The overall requirements for the TIS were laid out earlier in this section and can be phrased in terms of testing for infrastructure support of a family of mission threads of interest. These requirements can be formalized in terms of DEVS coupled models of mission threads and in terms of related measures of performance and effectiveness formulated as experimental frames. From this formal basis, the DEVS/SOA environment provides an infrastructure and tool set for implementing the TIS using DEVS test agents. An essential component of the tool set is the analytic capability to derive time windows for events and messages that determine the temporal usability of information to facilitate testing of MOPs for right place/right time arrival.

The DUNIP process requires that in parallel, a testing process for the implementation of the TIS be developed from the same formalized basis. We could of course, apply the TIS to itself – in

other words, using DEVS agents to monitor the activities of DEVS agents in their primary test federation application. However, taken literally, this could lead to an infinite regress in which another level is always needed to test the current level. Suitably restricted, this approach might be useful at a later stage of development after a first order level of confidence has been established. Early on, to get such confidence, the TIS should be tested on a simulated collaboration in a controlled setting where timings and behaviors of the simulated actors are controlled experimentally. For example, the collaboration model described in Exhibit 30 could be used where faults such as communication breakdowns and unresponsive elements can be introduced to establish anomalous test cases. The extent to which the TIS responds properly to both normal and abnormal test cases is a useful measure of its performance.

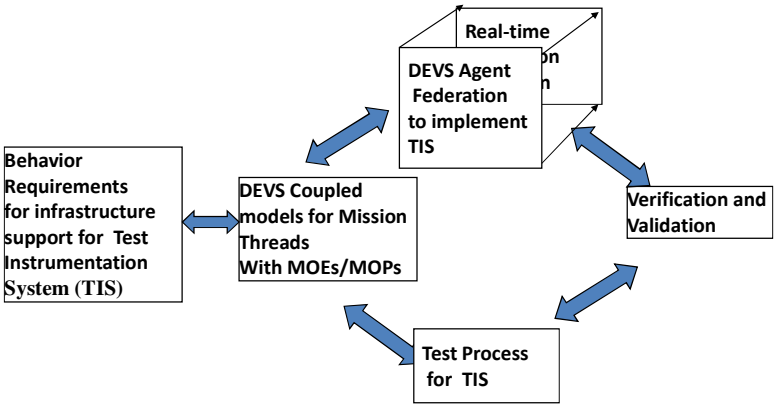

Exhibit 37: Bifurcated approach applied to design of Test Instrumentation System

10.6 Potential Issues and Drawbacks

Building upon and integrating earlier systems theoretic and architectural methodologies, DUNIP inherits many of the advantages that such methodologies afford and attempts to fill in the holes that they still leave. However, as with the methodologies it draws upon, and is inspired by, there are potential issues and drawbacks that may be expected to emerge in its application. Predictably, the current culture of system development still places more incentive on quickly finishing a project for an incremental development rather than on spending the extra resources needed to assure the existence of benefits such as reusability and extensibility that only provide cost-justification in the longer term. Until cultural change takes place to place greater emphasis on the fruits of well-founded methodological work, there is a risk that adopting DUNIP will create a situation in which schedules are missed and costs overrun. Particularly, personnel employed in DUNIP-based development must be adequately trained on all its supporting intellectual and technological components – system theory, DEVS, object-based software architecture, etc. Such prior education and experience should be regarded as mandatory for DUNIP adoption and if not present, steps must be taken to provide the necessary training, education, and experience. Further, training in one foundational element, such as software architecture, alone does not reduce the risk, and may even intensify it – in the spirit of a "little

knowledge is a dangerous thing." Hopefully, books like the one in which this chapter resides and others such as those mentioned earlier ([ZEI00], [ZEI07)] will take their place in the curricula that provide education and training the emerging SoSE field.

Further, the complexity and quality assurance issues associated with the proposed methodology need to be mitigated with the development of appropriate tools and interfaces to simplify working with the methodology. The need for such complexity-reduction tools underlies the extended discussions of tools and interfaces that have been provided here. Further quality assurance demands provision of approaches to self-checking DUNIP and its supporting infrastructures. The discussion of the Test Instrumentation system and its verification and validation provides and example of approaches to such assurance.

Finally, there remain many issues to resolve in the manner in which the DUNIP methodology relates to the defense-mandated DODAF. The latter is a representational mechanism, not a methodology, and does not discuss how an integrated architecture should be constructed and evolved from existing systems. DUNIP offers an approach based on systems theory and supported by DEVS-based modeling and simulation to tackle integration and interoperability issues, but the integration with DODAF remains for future consideration.

11. SUMMARY

In this chapter we have taken the challenge of constructing a System of System (SoS) as one of designing an infrastructure to integrate existing systems as components, each with its own structure and behavior. The SoS can be specified in many available frameworks such as DoDAF, system theory, UML, or by using an integrative systems engineering-based framework such as DEVS. In this chapter, we have discussed the advantages of employing an M&S-integrated framework such as DEVS Unified Process (DUNIP) and its supporting DEVS/SOA infrastructure. We illustrated how 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. We have illustrated, with an SoS consisting of heterogeneous robotic sensors and decision components, how the integration infrastructure must support interoperability at syntactic, semantic and pragmatic levels to achieve the requisite interoperation. We discussed DoD's Global Information Grid (GIG) as providing an integration infrastructure for SoS in the context of constructing collaborations of web services using the Service Oriented Architecture (SOA). The DEVS Unified Process (DUNIP), in analogy to the Rational Unified Process based on UML, offers a process for integrated development and testing of systems that rests on the SOA infrastructure. The DUNIP perspective led us to formulate a methodology for testing any proposed SOA-based integration infrastructure, such as DISA's Net-Centric Enterprise Services. To support such a methodology we proposed a Test Instrumentation System (TIS) built upon the integrated infrastructure that employs DEVS Agents to perform simultaneous testing at the syntactic, semantic, and pragmatic levels. Clearly, the theory and methodology for such SoS development and testing are at their early stages. While one book as appeared on the subject, [ZEI07], we challenge the reader to explore the issues involved and come up with more incisive solutions that extend the very essence of systems engineering theory.

A reviewer's comments provide a well-stated summary of the chapter – it "the proposed methodology can be used for integrating heterogeneous constituents of an SoS and assessing their

real time interactions and interoperability. The proposed methodology encompasses the advantages of several interrelated concepts such as the systems theory; DEVSML and DEVS/SOA; M&S framework; and the model continuity concepts. Especially, since it separates models from their underlying simulators; enables real time execution and testing at multiple levels and over a wide rage of execution platforms; uses open standards; supports collaborative development; and has the potential to provide additional SoS architectural views."

REFERENCES

[ACI06] ACIMS software site: http://www.acims.arizona.edu/SOFTWARE/software.shtml Last accessed Nov 2006

[ATK04] K. Atkinson, "Modeling and Simulation Foundation for Capabilities Based Planning", Simulation Interoperability Workshop Spring 2004

[BAD05] 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

[BUC04] J.B. Buchheister, "Net-centric Test & Evaluation", Command and Control Research and Technology Symposium: The Power of Information Age Concepts and Technologies, 2004, available at: www.dodccrp.org/events/2004/CCRTS_San_Diego/CD/papers/208.pdf, last accessed October 2005

[BPMN] Business Process Modeling Notation (BPMN) www.bpmn.org

[BPEL] Business Process Execution Language (BPEL) http://en.wikipedia.org/wiki/BPEL

[CAR05] Carstairs, D.J., "Wanted: A New Test Approach for Military Net-Centric Operations", Guest Editorial, ITEA Journal, Volume 26, Number 3, October 2005

[CHA05] 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.

[CJC04] Chairman, JCS Instruction 3170.01D "Joint Capabilities Integration and Development System," 12 March 2004.

[CJC06] Chairman, JCS Instruction 6212.01D "Interoperability and Supportability of Information Technology and National Security Systems," March 8, 2006, 271

[DAH98] 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

[DAN04] F Dandashi, H. Ang, C. Bashioum, "Tailoring DoDAF to Support a Service Oriented Architecture", White Paper, Mitre Corp, 2004

[DIM07] M.J. DiMario SoSE Discussion Panel Introduction, 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

[DIM06] 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

[DML] DEVSML – A Web Service Demonstration http://150.135.218.205:8080/devsml/

[DOD03a] DoDAF Working Group , DoD Architecture Framework Ver. 1.0 Vol. III: Deskbook, DoD, Aug. 2003.

[DOD03b] DOD Instruction 5000.2 "Operation of the Defense Acquisition System," 12 May 2003.

[DOD04c] DoD Architecture Framework Working Group 2004, DOD Architecture Framework Ver. 1.0 Volume 1 Definitions and Guidelines, 9 February 2004, Washington, D.C.

[DOD05] DoD Metadata Registry and Clearinghouse http://www.xml.gov/presentations/fgm/dodregistry.htm

[DUN07] DUNIP: A Prototype Demonstration http://www.acims.arizona.edu/dunip/dunip.avi

[GEN06] GENETSCOPE(Beta Version) Software User's Manual, available from ACIMS center, University of Arizona.

[HUX03] X. Hu, B.P. Zeigler, S. Mittal, "Dynamic Configuration in DEVS Component-based Modeling and Simulation", SIMULATION: Transactions of the Society of Modeling and Simulation International, November 2003

[HUX04] X. Hu, and B.P. Zeigler, " Model Continuity in the Design of Dynamic Distributed Real-Time Systems", accepted by *IEEE Transactions On Systems, Man And Cybernetics— Part A: Systems And Humans*

[IDEF] IDEF Functional Modeling Method, http://www.idef.com/

[JITC] JITC reports for SCOPE command for year 2005, JITC, latest data as of Oct 2005.

[JLR07] Martin, JLR, Mittal, S., Zeigler, B.P., Manuel, J., "From UML Statecharts to DEVS State Machines using XML", IEEE/ACM conference on Multi-paradigm Modeling and Simulation, Nashville, September 2007

[Lee05] J. Lee, M. Choi, J. Jang. Y, Park, J. Jang,, B. Ko, "The Integrated Executable Architecture Model Development by Congruent Process, Methods and Tools", The Future of C2, 10th International Command and Control Research and Technology Symposium

[MOR04] J. Morganwalp and A. P. Sage, "Enterprise Architecture Measures of Effectiveness," *International Journal of Technology, Policy and Management,* vol. 4, pp. 81-94, 2004.

[MAK06] Mak, E., Automated Testing using XML and DEVS, Thesis, University of Arizona, http://www.acims.arizona.edu/PUBLICATIONS/PDF/Thesis_EMak.pdf

[MAK07] E Mak, S Mittal, MH Hwang, "Automating Link-16 Testing using DEVS and XML", Journal of Defense Modeling and Simulation, to appear

[MIT06a] Mittal, S. "Extending DoDAF to Allow DEVS-Based Modeling and Simulation", Special issue on DoDAF, Journal of Defense Modeling and Simulation JDMS, Vol 3, No. 2.

[Mit06b] Mittal.S., Mak, E. Nutaro, J.J., "DEVS-Based Dynamic Modeling & Simulation Reconfiguration using Enhanced DoDAF Design Process", special issue on DoDAF, Journal of Defense Modeling and Simulation, Dec 2006

[MIT05c] S. Mittal, B. P. Zeigler, "Dynamic Simulation Control with Queue Visualization", Summer Computer Simulation Conference, SCSC'05, Philadelphia, July 2005

[MIT03d] S. Mittal, B.P. Zeigler, "Modeling/Simulation Architecture for Autonomous Computing", Autonomic Computing Workshop: The Next Era of Computing, Tucson, January 2003.

[Mit07e] Mittal, S., Risco-Martin, J.L., Zeigler, B.P. "DEVSML: Automating DEVS Simulation over SOA using Transparent Simulators", DEVS Syposium, 2007

[Mit07f] Mittal, S., Risco-Martin, J.L., Zeigler, B.P. "DEVS-Based Web Services for Net-centric T&E", Summer Computer Simulation Conference, 2007

[MIT07g] Mittal, S, DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures, Ph. D. Dissertation, University of Arizona

[MIT07h] Mittal, S., Hwang, M.H., Zeigler, B.P., "FD-DEVS: An Implementation of W3C Schema for Finite Deterministic DEVS", in progress, Demo available at: http://www.u.arizona.edu/~saurabh/fddevs/FD-DEVS.html

[NSB00] Network-Centric Naval Forces - Overview: A Transition Strategy for Enhancing Operational Capabilities (2000) Naval Studies Board (NSB)].

[OMG] Object Modeling Group (OMG) www.omg.org

[SIAP] 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.

[SAG07] 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

[SAG01] A. P. Sage and C. D. Cuppan, "On the Systems Engineering and Management of Systems of Systems and Federation of Systems," Information Knowledge Systems Management, vol. 2, pp. 325 - 345, 2001.

[SAH07a] F. Sahin, P. Sridhar, B. Horan, V. Raghavan, M. Jamshidi, "System of Systems Approach to Threat Detection and Integration of Heterogeneous Independently Operable Systems," Proc. of IEEE Int Cont. Systems, Man and Cybernetics, 2007.

[SAH07b] F. Sahin, M. Jamshidi, P.Sridhar, "A Discrete Event XML based Simulation Framework for System of Systems Architecture", Proc. of IEEE Int Cont. on Systems of Systems Engineering, 2007.

[SAR00] Sarjoughian, H.S., B.P. Zeigler, "DEVS and HLA: Complimentary Paradigms for M&S?" Transactions of the SCS, (17), 4, pp. 187-197, 2000

[SAR01] H. Sarjoughian, B. Zeigler, and S. Hall, *A Layered Modeling and Simulation Architecture for Agent-Based System Development*, Proceedings of the IEEE 89 (2); 201-213, 2001

[TRAO] Traore M, Muxy A, Capturing the Dual Relationship between Simulation Models and Their Context, ` SIMPRA (Simulation Practice and Theory), published by Elsevier, 2004

[TOL03] Tolk, A., and Muguira, J.A. The Levels of Conceptual Interoperability Model (LCIM). Proceedings Fall Simulation Interoperability Workshop, 2003

[TOL03] A. Tolk, S. Solick, "Using the C4ISR Architecture Framework as a Tool to Facilitate V&V for Simulation Systems within the Military Application Domain", Simulation Interoperability Workshop, Spring 2003

{UML] Unified Modeling Language (UML), http://www.omg.org/technology/documents/formal/uml.htm

[Wad02] L.W Wagenhals, S. Haider, A.H.Levis, "Synthesizing Executable Models of Object Oriented Architectures", Workshop on Formal Methods Applied to Defense Systems, Adelaide, Australia, June 2002

[Weg02] Wegmann, A., "Strengthening MDA by Drawing from the Living Systems Theory", Workshop in Software Model Engineering, 2002

[WAY76] Wayne A Wymore, A Mathematical Theory of Systems Engineering: The Elements, Krieger, Huntington, NY, 1967.

[WAY92] Wayne A Wymore, Engineering Modeling and Design, (with Bill Chapman and A. Terry Bahill), CRC Press Inc., 1992.

[xDEVS] XDEVS web page: http://itis.cesfelipesegundo.com/~jlrisco/xdevs.html

[xHLA] HLA, https://www.dmso.mil/public/transition/hla/

[xMAT] MatLab Simulink, http://www.mathworks.com/products/simulink/

[xNS2] NS-2, http://www.isi.edu/nsnam/ns/

[xOMN] OMNET++, http://www.omnetpp.org/

[SRI03] Sridhar, P., Sheikh-Bahaei, S., Xia, S., Jamshidi, M., "Multi-agent Simulation using Discrete Event and Soft-Computing Methodologies", IEEE International Conference on Systems, Man and Cybernetics, 2003.

[SRI04] Sridhar, P., Jamshidi, M., "Discrete Event Modeling and Simulation: V-Lab", IEEE International Conference on Systems, Man and Cybernetics, 2004.

[SRI06] P. Sridhar, A. M. Asad, M. Jamshidi, "Hierarchical Data Aggregation in Spatially Correlated Distributed Sensor Networks", *IEEE Co-Sponsored World Automation Congress - International Symposium on Robotics and Applications, Budapest,* 2006

[SRI07] P. Sridhar, A. M. Asad, M. Jamshidi, "Hierarchical Aggregation and Intelligent Monitoring and Control in Fault-Tolerant Wireless Sensor Networks", *IEEE Systems Journal*, vol.1, pp. 38-54, September 2007.

{YLMAZ] L.Ylmaz and T.I. Oren, "A Conceptual Model for Reusable Simulations within a Model-Simulator-Context Framework Conference on Conceptual Modeling and Simulation" , Conceptual Models Conference, Genoa, Italy , October 28-31 2004

[ZACH] Carol O'Rourke, Neal Fishman, "Enterprise Architecture Using the Zachman Framework" by, and Warren Selkow. ISBN 0-619-06446-3 Published by Course Technology, 2003. (www.eabook.info)

[ZEI00] Zeigler, B. P., T. G. Kim, and H. Praehofer. (2000). Theory of Modeling and Simulation. New York, NY, Academic Press.

[ZEI05] Zeigler, B.P., Fulton, D., Hammonds P., and Nutaro, J. Framework for M&S–Based System Development and Testing In a Net-Centric Environment, ITEA Journal of Test and Evaluation, Vol. 26, No. 3, 21-34, 20

[Zei05b] BP Zeigler, S Mittal, "Enhancing DoDAF with DEVS-Based System Life-cycle Process", IEEE International Conference on Systems, Man and Cybernetics, Hawaii, October 2005

[ZEI07] 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.