Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds

MODEL-DRIVEN SYSTEMS ENGINEERING FOR NETCENTRIC SYSTEM OF SYSTEMS WITH DEVS UNIFIED PROCESS

Saurabh Mittal

Dunip Technologies
OH 45434 USA

José Luis Risco Martín

Universidad Complutense de Madrid
Madrid, SPAIN

ABSTRACT

Model-Based Systems Engineering (MBSE) employs model-based technologies and established systems engineering practices. Model-Driven Engineering (MDE) provides various concepts to automate model based practices using metamodeling. We describe the DEVS Unified Process (DUNIP) that aims to bring together MBSE and MDE as Model-driven Systems Engineering (MDSE) and apply it in a netcentric environment. We historically look at various model-based and model-driven flavors and suggest MDSE/DUNIP as one of the derived methodologies. We describe essential elements in DUNIP that facilitate integration with architecture solutions like Service Oriented Architecture (SOA), Event Driven Architectures (EDA), Systems Entity Structures (SES) ontology, and frameworks like Department of Defense Architecture Framework (DoDAF 2.0). We discuss systems requirement specifications, verification and validation, metamodeling, Domain Specific Languages (DSLs), and model transformation technologies as applicable in DUNIP. In this article, we discuss the features and contributions of DUNIP in netcentric system of systems engineering.

1 INTRODUCTION

Model-Based Systems Engineering (MBSE) employs model-based technologies in conjunction with the established systems engineering practices. The prime motivation of MBSE is to use model as the foundational element in systems engineering and ensure that model becomes the actual system. *Model-Driven Engineering (MDE)* provides many software tools and practices that automate the model development and transformation, thereby facilitating the usage of model as an actual software component. The critical difference between MBSE and MDE is that the former is based on General Systems theory. Model-driven Systems Engineering (MDSE) is hereby defined, as a discipline that applies MDE practices to MBSE paradigm. In this article, we will look at *DEVS Unified Process (DUNIP)* that is based on *Discrete EVent Systems (DEVS)* formalism, and its contribution to MDSE. We execute MDSE/DUNIP in a netcentric environment. DEVS formalism is based on General Systems theory, has mathematical rigor and has been in existence for over 40 years. Complex systems and netcentric system of systems pose a different kind of challenge altogether when it comes to application of MDSE. We will describe essential elements in DUNIP that facilitate integration with architecture solutions like *Service Oriented Architecture (SOA)*, *Event Driven Architectures (EDA)*, *Systems Entity Structures (SES)* ontology, complex dynamical systems and finally frameworks like *Department of Defense Architecture Framework (DoDAF)* that help define and develop netcentric systems. While all these approaches build a specific type of system, the usage of MBSE to deliver the complete solution is not unified, and is largely undertaken by graphical languages like the *Unified Modeling Language (UML)* and *Systems Modeling Language (SysML)*, both of which lack mathematical rigor. We take a look at such requirement specifications and how metamodeling, *Domain Specific Languages (DSLs)* and model transformation technologies provide mechanisms to integrate with larger systems requirements. It places current M&S methodologies at the forefront of a new paradigm i.e. MDSE with DUNIP, where both modeling and simulation technologies coexist and contribute to the definition and development of novel netcentric M&S systems. Further, one of the critical pieces of systems engineering is the satisfaction of requirements across the entire systems life cycle. The application of *Verification and Validation (V&V)* and *Testing and Evaluation (T&E)* to netcentric complex systems present new challenges. Requirements traceability in such an engineering effort is a critical game changer. Within DUNIP, the inclusion of DSLs with various model-to-DEVS transformations eventually cast models as formal system components by executing through the netcentric *DEVS Virtual Machine (DEVSVM)* (Mittal and Martin 2013). Such models can be easily integrated with a local system or a larger distributed netcentric system offering solutions within the MDSE paradigm. In this article, we will take a look at some of the above issues and discuss the features and the contributions of DUNIP in netcentric system of systems engineering .

The rest of the paper is organized as follows. Section 2 presents historical background. It also describes various underlying technologies utilized to realize MDSE in a netcentric environment. Section 3 provides an overview on DUNIP and elaborates on the realization of MDSE with DUNIP. Section 4 describes netcentric SoS engineering with SOA, EDA and DEVS-based systems. Finally, Section 5 presents conclusions.

2 THE FOUNDATION

This section will lay the foundation for realizing the MBSE vision in the development of a netcentric SoS with DEVS Unified Process. We will describe the technologies, processes and various other considerations that are used in SoS engineering. This section provides an overview of the foundational work and the authors take responsibility of presenting the salient aspects of three decades of work as it is very difficult to summarize them as a subsection in this paper when entire books have been written on these subjects. However, it is essential to include them in this article.

2.1 Model-based and Model-driven flavors

The 'model-based (MB)' and 'model-driven (MD)' terms and initials have been used in a variety of system and software related acronyms, such as MBD, MDSD, MDD, MDA, MBSE, MDE and many others. Although there is a consensus that these approaches suggest the systematic use of models as the primary means of a process and facilitate the use of domain specific languages, there is not a common understanding of the terminology (Cetinkaya et al. 2013). In this section, the definitions of the frequently referred acronyms and the objectives of those approaches are presented..

a. **MBE:** Model-based Engineering (MBE) originated in the 1970's in parallel with the evolution of the Computer-Aided Design (CAD) and Model-Based Design (MBD) techniques. The main goal in MBE was to support the system development process during the design, integration, validation, verification, testing, documentation and maintenance stages. A decade long pioneering work by Wymore and Zeigler, based on the foundations of mathematical systems theory (Wymore 1967) and model-based engineering was introduced in the this era (Wymore 1976, Zeigler 1976, Zeigler 1984).
b. **MBSE:** In systems engineering, the application of the MBE principles is called as Model Based Systems Engineering (MBSE) (Zeigler 1984, Wymore 1984, Zeigler and Chi 1993, Wymore 1993, Oliver, Andary and Frisch 2009). MBSE provides the required insight in the analysis and design phases, enhances better communications between the different participants and enables effective management of the system complexity. A summary of DEVS-based integrated development environments is available in (Zeigler and Sarjoughian, 2012).
c. **MDE:** Model-driven engineering (MDE) is a system development approach that uses models to support various stages of the development life cycle (Atkinson and Kuhne 2003, Schmidt 2006) and can be seen as a subset of MBE. MDE relies on technologies to automate model transformations thereby increasing productivity within MBE. It produces well-structured and maintainable systems because of its focus on formally defined models, metamodels, and meta-metamodels**.**
d. **MDA:** Model-Driven Architecture (MDA) is a software design and development approach that provides a set of guidelines for specifying and structuring models (OMG 2003) MDA relies on the Meta

Object Facility (MOF) (OMG 2006)to integrate the modeling steps and provide the model transformations. MDA prescribes the use of metamodels and meta-metamodels for specifying the modeling languages without any necessity to be domain specific.

e. **MDD or MDSD:** The application of the MDE principles in software engineering is called MDD (Model Driven Development) or MDSD (Model Driven Software Development) (Volter, et al. 2006). The modern era of MDD started in the early 1990s and now offers a notable range of methods and tools. Different specifications such as MDA (OMG 2003), MIC (ISIS 1997), Eclipse Modeling Project (Eclipse 2008) and Microsoft Software Factories (Microsoft 2005) are some of the conceptual applications of MDD principles.
f. **MIC:** MIC (Model Integrated Computing) refines the MDD approaches and provides an open integration framework to support formal analysis tools, verification techniques and model transformations in the development process (ISIS 1997). MIC allows the synthesis of application programs from models by using customized Model Integrated Program Synthesis (MIPS) environments (e.g., Generic Modeling Environment [GME]). The meta-level of MIC provides metamodeling languages, metamodels, metamodeling environments and meta-generators for creating domain specific tool chains on the MIPS level.

As a result, MB/MD approaches place models at the core of an entire system/software development process and try to increase the system specification quality. An MB/MD expert can easily think about combining different approaches and tools to incorporate in an MDSE process. One other fundamental concept that is important to understand is that the pioneering work (Oren and Zeigler 2012) in the area of MBE and MBSE involved the development of simulation technologies as an integral part of model execution framework. No MBSE will be complete without the development of a simulator that executes the model. With the advancement of software engineering practices, models development methodologies integrated very closely with the software code and technologies like UML and SysML pushed the envelope with graphical modeling notations. However, the execution of model through a simulator is a complete discipline in itself and till date executable UML/SysML are incomplete as they lack theoretical systems foundation to begin with (Palmer 2004, Mittal 2006). At best, they serve as DSLs for a community who want to describe systems who may lack formal systems engineering training. However, in the past decade there have been many independent efforts from the M&S/systems engineering community to transform UML/SysML to formal systems engineering frameworks such as DEVS and align the graphical expressiveness with foundational systems theory. Further, the engineering of high performance simulators requires advanced distributed computing algorithms and capabilities. MDA, MDD/MDSD and MIC deal with integrating models directly into the software resulting in a software that needs to go through various other model-checking and verification methodologies. This need of bringing together advanced model engineering technologies such as MDA with M&S software has been acknowledged earlier (Tolk 2002). The model-verification technologies in MDA/MDD/MDSD bear a strong software engineering flavor. However, the incorporation of systems theory for an integrative effort was partly lacking as MDA, UML, XML and other Internet-based standards pursued it from a purely software engineering approach.

MBE and MBSE utilize the systems verification and validation methodologies to the model development process relating back to the system requirements for systems test and evaluation. MDE is one area that serves both the software and systems engineering as it is domain independent. MDSE gives MB/MBSE various model engineering, transformation and tools from MDE that speed up model development. The development of various editors based on MDE concepts involve advanced software engineering. These technologies can be visualized on a continuum as shown in Figure 1. Application of such advanced software practices for modeling and simulation area has recently been accomplished in other parallel efforts like MDD4MS (Cetinkaya, Verbraeck, and Seck 2011) and executing DEVS models on a cloud by Zeigler and Sarjoughian (2012). MDD4MS utilized MDE to a larger extent but is more inclined towards software engineering. The approach by Zeigler and Sarjoughian is focused towards fundamental systems engineering and its extension to cloud-based environments for collaborative modeling. The usage of MDE in Zeigler and Sarjoughian's approach is not explicit. Between these two solutions that use advanced software engineering practices, DUNIP is positioned as an MDSE that employs both the systems engineering and MDE paradigm in an agile manner. To understand MDE, once has to understand the concept of metamodeling and domain specific languages.

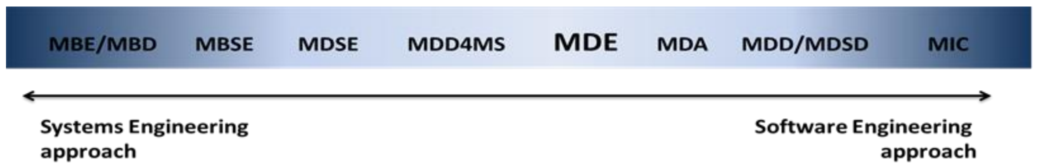

Figure 1: MB/MD technologies with Systems/Software engineering approaches

2.2 Metamodeling

Metamodeling is the process of complete and precise specification of a metamodel. A metamodel is a definition of a modeling language in the form of a model. In other words, a metamodel is itself a model and has to be defined in a modeling language, called a formalism language and it is at this level a domain specific semantics are embedded formally in a language, also defined with a metamodel. The metamodel of the formalism language is called a meta-metamodel. At this meta-meta level, various language editors are defined that allow the creation of multiple domain specific languages or formalisms. Most approaches implementing MDE define a three level metamodeling stack for model (M1), metamodel (M2), and meta-metamodel (M3). At the lowest level, M0 is the end user that creates instances of the model for its application usage. Figure 2 shows the metamodeling pattern.

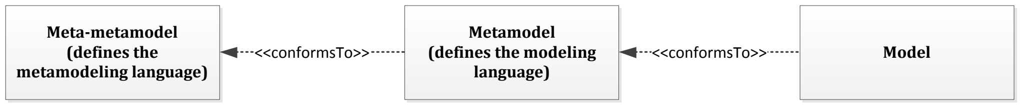
Meta-metamodel (defines the metamodeling language)
<<conformsTo>>
Metamodel (defines the modeling language)
<<conformsTo>>
Model

Figure 2: Metamodeling pattern in MDE

The general application of metamodeling is performed with four levels, which was first described in the UML (Unified Modeling Language) Specification by OMG (Object Management Group) (OMG 1999). The MOF (OMG 2006) is the de-facto metamodeling language in OMG specifications for the M3 level.

2.3 Domain Specific Languages

In MDE, a DSL (Domain Specific Language) is a language designed for a particular problem domain and is specified at the M2 level. The term "domain-specific" is generally used for expressing different application domains such as logistics, health care, airports and container terminals. The abstractions defined in a domain specific language map directly to the problem domain concepts that it chooses to represent. A DSL is meant to hide the complexities of the computational domain and highlight the complexities of the domain it is designed for (Ghosh 2010). A DSL is specified using a metamodel. The metamodeling process is called as Domain Specific Modeling (DSM). Now, having given the general-purpose definition of a domain specific language, we inherently assume that the domain modeling language is an executable piece of code that is semantically anchored to a computational programming language. This assumption is based on the premise that the modeling language and its execution platform are considered together when we talk about domain specific modeling. As stated earlier, MDE promotes the use of DSLs and automated transformations to build executable code.

2.4 Theory of Modeling & Simulation

One of the most important theories on modeling and simulation was pioneered by Zeigler (1976). It defines the elements and systems specifications for model-driven simulation-based systems. The discrete event systems (DEVS) theory is made of two orthogonal concepts:

1. *Hierarchy of Systems Specification:* there are 5 levels that describe how systems behave;
2. *System Specification Formalisms:* these incorporate various modeling styles, such as continuous or discrete.

Systems theory distinguishes between system structure (how the system is constituted internally) and system behavior (how the system manifests externally). Understanding the system structure allows us to deduce its behavior. Systems theory is *closed under composition* in that the structure and behavior of a composition of systems can be expressed in original system theory terms. This is the foundation of modular systems that have defined input and output interfaces through which all interaction with the environment occurs. Such modular systems are coupled together to form larger ones leading to hierarchical construction. The hierarchy of systems specification (Zeigler 1976) based on Levels of system specifications (Klir 1969) has 5 levels as shown in Table 1. It also relates various concepts as defined in the theory of modeling and simulation (Zeigler, Praehofer and Kim 2000).

Table 1: Hierarchy of System specifications

Level	Name	System Specification at this level	Elements from the Framework for M&S	Verification and Validation
4	Coupled Systems	Systems built from component systems with a coupling recipe	Model, Simulator, Experimental Frame	Structural Validity, simulator correctness
3	I/O System Structure	System with state and transitions to generate the behavior	Model, Simulator, Experimental Frame	
2	I/O function	Collection of input/output pairs partitioned according to initial state	Model, Source System	Predictive Validity
1	I/O behavior	Collection of input/output pairs from external black-box view	Model, Source System	Replicative Validity
0	I/O frame	Input and output variables and ports together with values over a time base	Source System	

Description of alternate system specification formalisms such Differential Equation System Specification (DESS), Discrete Time System Specification (DTSS), and Quantized Systems are briefly described in (Mittal and Martin 2013), Chapter 3 and can be found in much greater detail in (Zeigler, Praehofer and Kim 2000). While DESS and DTSS as their names suggest are self-explanatory, Quantized DEVS warrants a definition. Quantization is a process for representing and simulating continuous systems as an alternative to the more conventional time axis. It is built on *threshold crossing* model. While *discretization* leads to DTSS, *quantization* leads to discrete event systems. The universality of DEVS formalism allows specification of hybrid systems.

3 MDSE WITH DEVS UNIFIED PROCESS

The DEVS Unified Process is based on an *open systems* concept. An *open system* is a dynamical system that can exchange energy, material and information with the outside world through its reconfigurable interfaces. A netcentric system tends to an *open* system. Much of the *open system* development hinges on the variable structure capability within a component-based system. A netcentric SoS provides a dynamic environment where the dynamic composable landscape (using publish/subscribe mechanisms) is analogous to a variable structure system.

DUNIP is the consummation of how DEVS can be applied to System of Systems design and analysis in full systems engineering life cycle setup (Mittal 2007). DUNIP is not a single concept but an integration of various concepts that have been developed over the years in DEVS research. These concepts have now evolved into an integrated process that facilitates systems modeling and simulation. The understated objective of DUNIP is to incorporate discrete event systems theory as the binding factor at all the phases of this development process. Figure 3 illustrates the DEVS Unified Process. Figure 4 shows the inherent spiral nature in DUNIP and the agility it brings by way of component-based engineering process. The important concepts in DUNIP as described in (Mittal and Martin 2013) are listed below:

1. Requirements specification using Domain Specific Languages (DSLs). Such DSLs include UML/SysML, Business Process Modeling Notation (BPMN), Department of Defense Architecture Framework (DoDAF) Version 2.0, and many other customized DSLs.
2. Platform Independent modeling (PIM) at lower levels of systems specification using DEVS DSL as specified in DEVSML stack (Section 3.1)
3. Model Structures at higher level of System resolution using System Entity Structures (SES)
4. Platform Specific Modeling that takes PIM and anchor it with DEVS semantics and execution environment
5. Automated Test Model generation using DEVS PIMs
6. Netcentric execution that allows the deployment of a netcentric DEVS VM in a distributed environment
7. Interfacing of models with real-time systems where the model becomes the actual systems component
8. Verification and Validation at every level of system specification and lifecycle development.

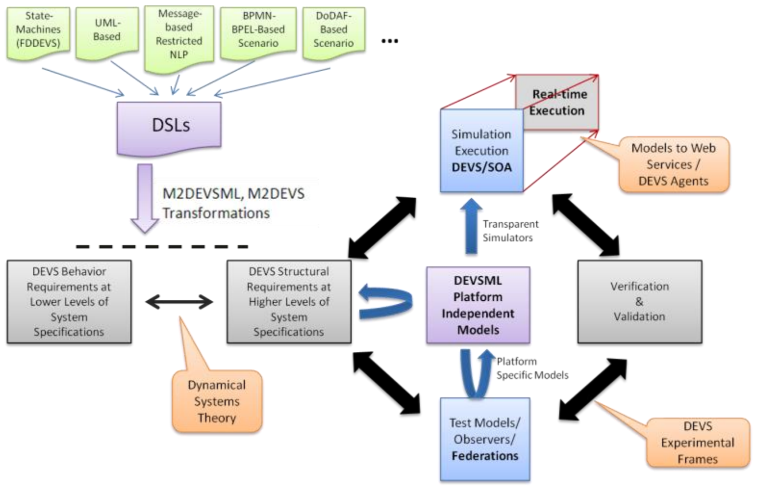
State-Machines (FDDEVS)
UML-Based
Message-based Restricted NLP
BPMN-BPEL-Based Scenario
DoDAF-Based Scenario
...
DSLs
M2DEVSML, M2DEVS Transformations
DEVS Behavior Requirements at Lower Levels of System Specifications
DEVS Structural Requirements at Higher Levels of System Specifications
Dynamical Systems Theory
Simulation Execution DEVS/SOA
Real-time Execution
Models to Web Services / DEVS Agents
Transparent Simulators
DEVSML Platform Independent Models
Platform Specific Models
Verification & Validation
Test Models/ Observers/ Federations
DEVS Experimental Frames

Figure 3: The DEVS Unified Process

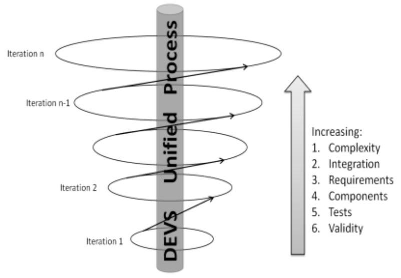

Figure 4: Agile/Spiral nature of DUNIP

3.1 DEVSML and the Netcentric DEVS Virtual Machine (DEVSVM)

The application of both MDE and MBSE in DUNIP is best described by the DEVSML framework. Decoupling the model from the simulation platform has many benefits as it allows the modeler to construct models in a platform of his choice. The ability to execute DEVS models in multiple platforms (Windows, linux, mac, etc) and languages (C++, Java, C#, Scheme, etc.) has already been achieved. Figure 5 shows the relationship between the DEVS modeling and the DEVS simulation layers through a 'simulation relation'.

It also shows that the design of DEVS models was dependent on the language where the underlying assumption always has been "everything is an Object". In the new framework, we migrate from this concept to "everything is a model", including the transformations as defined in MDE. With this paradigm shift we find ourselves in the world of concepts and abstraction that are very specific to the domain which is 'to be modeled'. This gives rise the discipline of domain specific modeling and the metamodels that help specify these domain models. Choosing DEVS as the execution framework is recommended due to its rich history of model specification, simulator verification and its adherence to systems theory.

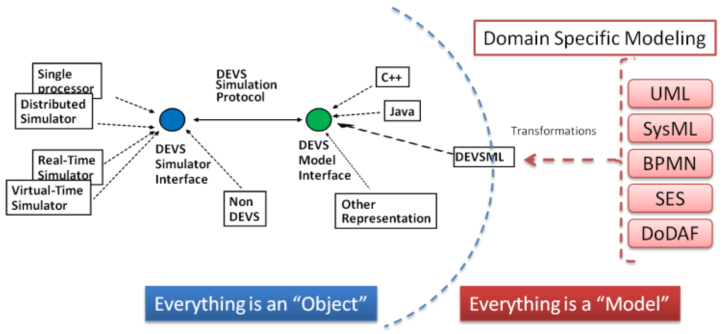

Figure 5. Integrating theory of modeling and simulation framework with MDE

This ability provides a solution to scale, integration and interoperability as described in (Mittal 2007, Mittal, Martin and Zeigler 2007, Zeigler, Mittal and Hu 2008, Mittal and Zeigler 2009, Martin, et al. 2009, Mittal, Zeigler and Risco-Martin 2010). Having a process to transform any DSL to DEVS components, especially to the DEVSML PIM, then has obvious advantages. The framework that takes DSLs to DEVS M&S netcentric environment is called DEVSML stack (Figure 6).

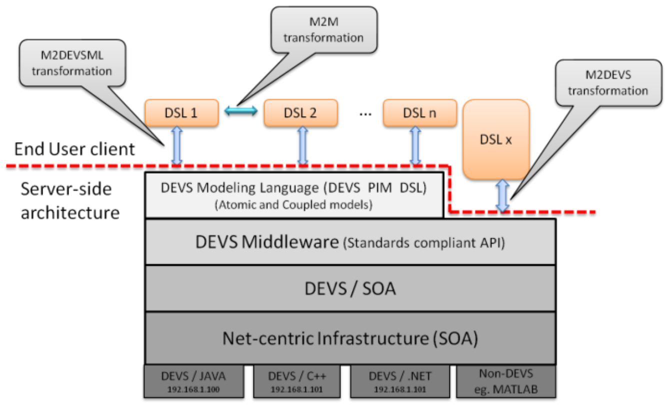
M2DEVSML transformation
M2M transformation
M2DEVS transformation
DSL 1
DSL 2
...
DSL n
DSL x
End User client
Server-side architecture
DEVS Modeling Language (DEVS PIM DSL) (Atomic and Coupled models)
DEVS Middleware (Standards compliant API)
DEVS / SOA
Net-centric Infrastructure (SOA)
DEVS / JAVA 192.168.1.100
DEVS / C++ 192.168.1.101
DEVS / .NET 192.168.1.101
Non-DEVS eg. MATLAB

Figure 6. DEVSML 2.0 stack showing DEVS modeling language and various transformations

The DEVSML stack (Mittal, Martin and Zeigler 2007) was refined as a part of Air Force Research Laboratory's research efforts by Mittal & Douglass (2011, 2012) as DEVSML 2.0 stack (Figure 6). The middleware in DEVSML stack is based on DEVS M&S Standards compliant API (under evaluation) and interfaces with a netcentric DEVS simulation platform such as a SOA that offers platform transparency. The idea of including other DSLs and the following transformations at the top layer of the stack is a major addition om DEVSML 2.0 that brings in MDE concepts with the DEVS M&S framework:

1. Model-to-Model (M2M)
2. Model-to-DEVSML (M2DEVSML)
3. Model-to-DEVS (M2DEVS)

3.2 DEVS Unified Process and MB Flavors

DEVSML and DUNIP are focused towards interoperability at the application level, specifically, at the modeling level and hiding the simulator engine as a whole. Our vision and solution development is along the lines of Model-as-a-Service (MaaS), Simulation-as-a-Service (SimaaS), DEVS-as-a-Service (DevaaS) and ultimately, System-as-a-Service (SysaaS). We are focused towards using XML just as a communication middleware, as used in SOAP, for existing DEVS models, but not as complete solution in itself. We would like the user or designer to code the behavior in any of the programming languages, ideally a DSL of his choice and let the DEVSML 2.0 stack develop the transformations. The DEVS/SOA architecture is responsible for taking a DSL or a coupled DEVSML model with the associated transformations and delivering us with an executable model that can be simulated on any DEVS platform. The realization of netcentric DEVS has the following pieces:

1. DEVSML Stack: the central concept
2. Distributed simulation using SOA
3. Netcentric DEVS VM (both client and server)
4. Design, development and deployment of netcentric systems with DEVS

The user can integrate his model from models stored in any web repository, whether it contained public models of legacy systems or proprietary standardized models will provide more benefit to the industry as well as to the user, thereby truly realizing the model-based paradigm. Table 2 contrasts various model-based and model-driven flavors with the MDSE/DUNIP methodology.

Table 2. MBSE/DUNIP in comparison with other model-based flavors

	System/Software Engineering approaches							
Features	**MBE**	**MBSE**	**MDSE / DUNIP**	**MDD4MS**	**MDE**	**MDD / MDSD**	**MDA**	**MIC**
Use of DSLs	Y	Y	Y	Y	Y	Y	Y	Y
Alignment with Systems theory	Y	Y	Y	-	-	-	-	-
DSL representation with metamodeling	-	-	Y	Y	Y	Y	Y	Y
Guidance for model transformations	-	-	Y	Y	Y	Y	Y	Y
Support for component reusability	Y	Y	Y	Y	-	-	-	Y
Code generation/execution	Y	Y	Y	Y	-	Y	Y	Y
Code deployment mechanisms	Y	Y	Y	-	-	Y	-	Y
Tool support for overall process	-	Y	Y	Y	Y	Y	-	Y
Applicable to all domains	Y	Sys. Engg.	Sys. Engg.	Y	Y	Soft. Engg.	Soft. Engg.	Soft. Engg.

4 NETCENTRIC SOS ENGINEERING WITH SOA, EDA AND M&S SYSTEMS

Service Oriented Architecture (SOA) enables orchestrating web services to execute processes for efficient information reuse, integration, collaboration and cost-sharing. As an example, the Department of Defense's (DoD) grand vision is the Global Information Grid that is founded on SOA infrastructure. The SOA infrastructure is to be based on a small set of capabilities known as Core Enterprise Services (CES) whose use is mandated to enable interoperability and increased information sharing within and across Mission Areas, such as the Warfighter domain, Business processes, Defense Intelligence, and so on (CIO 2007). Net-Centric Enterprise Services (NCES) is DoD's implementation of its Data Strategy over the GIG. NCES provide

SOA infrastructure capabilities such as service and metadata registries, service discovery, user authentication, machine-to-machine messaging, service management, orchestration, and service governance. The latest version of Department of Defense Architecture Framework (Version 2.0) mandated by DoD as the common denominator for DoD wide systems sharing and integration has adopted the model-based paradigm to bring architectures executable on GIG.

In a SoS, systems and/or subsystems often interact with each other because of interoperability or lack of thereof and integration of the SoS. In a netcentric SoS, such systems are integrated using standards-based protocols & middleware (e.g. XML, SOAP, etc.) and interact through service interfaces specified by Web Service Description Language (WSDL). The resulting architecture is a SOA. A system architecture where the constituent systems communicate using events/messages is called an Event driven Architecture (EDA). SOA is a great enabler for EDA.

An EDA is a stateless system that is driven by events and is composed of the following functional components: a) Event producer, b) Event consumer/listener, c) Event processor, d) Event Reaction, and e) a messaging backbone. An event producer generates a pragmatic event and put it on the messaging backbone. The event consumer picks up the pragmatic event that is usable by the event consumer. The consumer forwards it to the event processor for action on the event which generates either an automated or human reaction in the form of another event that may or may not be put on the messaging backbone. In an EDA, the interoperability at the syntactic level between a producer, consumer, and the processor is taken for granted, as the adoption of open standards and selection of XML as the preferred mode of communication. An EDA is granular at the event level and is decentralized. EDA uses commonly accessible messaging backbone, such as an Enterprise Service Bus (ESB) as well as adapters or middleware to transport messages/events. Finally, since EDA is stateless, the system's state or context must be encapsulated within the events. In an EDA, the event processing is of three types: a) **Simple event processing:** each event is processed exclusively and EDA may not generate an event reaction; b) **Event stream processing**: events have a temporal nature and multiple events bearing correlation may elicit a reaction, and lastly, **Complex event processing (CEP)**: the processors attend to multiple event streams on different time scales, thereby logically correlating them into multiple meaningful reactions. At this level, CEP almost symbolizes pattern matching on information sets as addressed in Artificial Intelligence (AI) and cognitive science literature. An EDA that implements CEP is inherently complex, dynamic and implicit. The parallels between EDA and DEVS can be best understood by mapping with DEVS levels of system specification as shown in Table 3.

Table 3**.** Mapping EDA constructs to DEVS levels of system specifications.

Level	**Name**	**EDA**
4	Coupled systems	*Does not exist. There is no containment to specify system hierarchy.*
3	I/O System	*Does not exist*
2	I/O Function	Complex event processing
1	I/O Behavior	Event stream processing
0	I/O Frame	Simple Event processing

An EDA built on DEVS atomics and coupled components adds structure to the underlying system, thereby managing state and hierarchical components that are abstracted behind the formal event descriptions (Mittal and Martin 2013). An EDA is real-time by default because there is no framework to manage abstract time, unlike DEVS. With DEVS M&S framework augmenting the EDA paradigm, an EDA can be used for large scale simulation systems that can run in logical time. Figure 7 shows an EDA and DEVS system together. DEVS virtual machines communicate with each other as per the DEVSML stack and can also put message events in the event cloud for EDA to act on them.

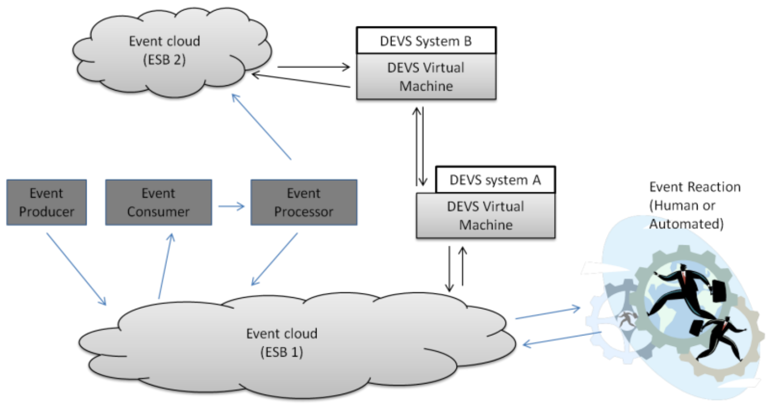

Figure 7**.** EDA functional components with netcentric DEVS systems

5 DISCUSSION AND CONCLUSIONS

MBE and MBSE were conceptualized in late 1970s and have been in application in systems engineering for the last four decades. As the field of object-oriented software engineering advanced, model-based concepts borrowed from these two paradigms were applied to this area resulting in other paradigms such as MDD, MDSD and MIC. Between these two engineering disciplines, MDE emerged that brought various automated model transformation technologies that expedite model development and execution and benefited both engineering disciplines. DEVS, since its inception, pioneered MBSE and is largely employed for complex dynamical systems engineering. DEVS Unified Process (DUNIP), a technological extension of DEVS systems engineering methodology, incorporates the concepts of MDE with DEVS systems engineering practices. The incorporation of advanced model tooling and the latest in model transformation technologies, a pure software engineering product, pushed by likes of Eclipse foundation, is facilitated by the DEVSML stack. The layered structure of the DEVSML stack derives directly from the DEVS theory of modeling and simulation that categorically separates the model and the simulator and further extends it towards the end-user that designs the model in a language of his choice, thereby freeing the model from the shackles of the hard-core programming languages (C++, Java, etc.). Such choices are reflected in various domain specific languages (DSLs), both textual and graphical (e.g., UML, SysML, BPMN, DoDAF, SES, etc.) that can now execute on a formal M&S framework transparently. This allows for model interoperability with transparent simulator execution, deployed locally or in netcentric environment.

A Netcentric environment is a distributed computing environment where computing devices share information through commonly accepted standards such as XML, SOAP, HTTP, WSDL and many others. SOA is a netcentric system architecture where devices share and communicate services through a defined interface specified by a WSDL. EDA is another netcentric system architecture where devices share message/events when they communicate with each other and has many other architectural components such as an event processor (often called event-cloud) and human/agent-in-the-loop. SOA is a key enabler for EDA as it enforces message/event structure and standards. DUNIP with its netcentric DEVS VM in the DEVSML stack, allows transparent execution of a DEVS model in a netcentric environment and is easily integrated with message/events exchanged on Enterprise Service Buses in SOA/EDA resulting in a netcentric M&S system that can interact with live web services.

A system of system (SoS) displays emergent behavior. A netcentric SoS is no different. Further, with a human-in-the-loop, a netcentric SoS can be characterized as a CAS (Mittal and Martin 2013) that evolves as it continues to interact, share and adapt with other components (including humans) in the system. While we can identify CAS behavior when we see one, engineering CAS is quite difficult primarily because there is no universal theory on CAS engineering analysis that integrates M&S as a part of its design cycle. Zeigler (2004) stated discrete event abstraction be used for modeling CAS and Mittal (2012) suggested extensions to DEVS that are needed to model CAS. MDSE with DUNIP brings formal M&S to the forefront of netcentric SoS development and engineering coupled with agile systems/software development lifecycle. It integrates various state-of-the-art model development technologies with advanced enterprise software engineering frameworks that define SOAs and EDAs. MDSE with DUNIP also acknowledges the role of the end-user and its contribution across the entire systems development lifecycle, beginning from his collaboration in the development of a DSL, to transformations and semantic mapping with DEVS theory, and ultimately to the actual use of the model as a systems component after its deployment in a netcentric environment. Various applications of DUNIP are available in the book (Mittal and Martin 2013).

REFERENCES

Atkinson, C., and T. Kuhne. 2003. "Model-driven development: a metamodeling foundation." *IEEE Software* 20 (5): 36-41.

Cetinkaya, D., A. Verbraeck, and M. D. Seck. 2011. "MDD4MS: A model drive development framework for modeling and simulation." *Proceedings of the Summer Computer Simulation Conference.* The Hague, Netherlands. 113-121.

Cetinkaya, D., S. Mittal, A. Verbraeck, and M. D. Seck. 2013. "Model Driven Engineering and its Application in Modeling and Simulation", in *Netcentric System of Systems Engineering with DEVS Unified Process*, by S. Mittal and J. L. R. Martin, 221-248, CRC Press

CIO, DoD. 2007. "Department of Defense Architectural Vision, Version 1.0." June. http://www.defenselink.mil/cio-nii/docs/GIGArchVision.pdf.

Eclipse. 2008. *Generic Eclipse Modeling System (GEMS).* Accessed April 24, 2012. http://www.eclipse.org/gmt/gems.

Ghosh, D. 2010. *DSLs in Action.* Manning Publications.

ISIS. 1997. *Model Integrated Computing (MIC).* Accessed May 10, 2012. http://www.isis.vanderbilt.edu/research/MIC.

Klir, G. 1969. *An approach to general systems theory*. New York: Van Nostrand Reinhold.

Martin, Jose Luis Risco, A. Moreno, J. Aranda, and J. M. Cruz. 2009. "Interoperability between DEVS and non-DEVS models using DEVS/SOA." *Spring simulation Multiconference.* San Diego, CA.

Microsoft. 2005. *Software Factories.* Accessed May 13, 2012. http://msdn.microsoft.com/en-us/library/bb977473.

Mittal, S. 2006. "Extending DoDAF to Allow DEVS-based Modeling and Simulation." *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* 3 (2).

Mittal, S., J. L. R. Martin, and B. P. Zeigler. 2007. "DEVSML: Automating DEVS Simulation over SOA using Transparent Simulators." *DEVS Symposium, Spring Simulation Multiconference.* Norfolk, VA.

Mittal, S., J. L. R. Martin, and B. P. Zeigler. 2007. "DEVS-Based Web services for Net-centric T&E." *Summer Computer Simulation Conference.* San Diego, CA.

Mittal, S. 2007. *DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures.* Tucson, AZ: PhD Dissertation, University of Arizona.

Mittal, S., J. L. R. Martin, and B.P. Zeigler. 2009. "DEVS/SOA: A Cross-platform Framework for Net-centric Modeling and Simulation in DEVS Unified Process." *Transactions of SCS* 85 (7).

Mittal, S., B. P. Zeigler, and J. L. R. Martin. 2010. "Implementation of Formal Standard for Interoperability in M&S/System of Systems Integration with DEVS/SOA." *International Command and Control Journal.*

Mittal, S., and S. A. Douglass. 2011. "From Domain Specific Languages to DEVS Components: Application to Cognitive M&S." *Proceedings of the Workshop on Model-driven Approaches for Simulation Engineering -- Spring Simulation Multiconference.* Boston, MA.

Mittal, S. 2012. "Emergence in Stigmergic and Complex Adaptive Systems: A Formal Discrete Event Systems perspective." *Journal of Cognitive Systems Research, Special Issue on Stigmergy.*

Mittal, S., and S. A. Douglass. 2012. "DEVSML 2.0: The Language and the Stack." *Symposium on Theory of Modeling and Simulation, Spring Simulation Multiconference.* Orlando, FL: SCS.

Mittal, S., and J. L. R. Martin. 2013. *Netcentric System of Systems with DEVS Unified Process.* Boca Raton, FL: CRC Press.

Oliver, D. W., J. F. Andary, and H. Frisch. 2009. "Model-based Systems Engineering." In *Handbook of Systems Engineering and Management*, by A.P. Sage and W.B. Rouse, 1361-1400. John Wiley & Sons.

OMG.1999. *UML Specification Version 1.3.*

—.2003. *Model Driven Architecture (MDA) Guide Version 1.0.1.* http://www.omg.org/mda/specs.htm.

—.2006. *Meta Object Facility (MOF) Core Specification, Version 2.0.* http://www.omg.org/spec/MOF/2.0.

Oren, T., and B. P. Zeigler. 2012. "System theoretic foundations of modeling and simulation: a historic perspective and the legacy of A Wayne Wymore." *Transactions of SCS* 88 (9): 1033-1046.

Palmer, K. 2004. "A Critique of SysML from the point of view of SysML." *holonomic.info.* Accessed May 22, 2013. http://holonomic.info/sm101a03.pdf.

Schmidt, D. C. 2006. "Model-driven Engineering." *IEEE Computer* 39 (2): 25-31. doi:9.1109/MC.2006.58.

Tolk, A. 2002. "Avoiding another green elephant - a proposal for the next generation HLA based on the Model Driven Architecture." *Fall Simulation Interoperability Workshop.* Orlando, FL.

Volter, M., T. Stahl, J. Bettin, A. Haase, and S. Helsen. 2006. *Model-driven software development: technology, engineering, management.* John Wiley & Sons.

Wymore, W. 1967. *A Mathematical Theory of Systems Engineering: The Elements*. New York: John Wiley

—. 1976. *Systems Engineering Methodology for Interdisciplinary Teams.* New York: John Wiley & Sons.

—. 1984. "The Tricotyledon Theory of System Design." In *Simulation and Model-based Methodologies: An Integrative View*, by T.I. Oren, B.P. Zeigler and M.S. Elzas, 119-132. New York: Springer-Verlag.

—. 1993. *Model-Based Systems Engineering.* Boca Raton, FL: CRC Press.

Zeigler, B.P. 1976. *Theory of modeling and simulation.* Wiley Interscience.

—. 1984. *Multifaceted Modelling and Discrete Event Simulation.* London, UK: Academic Press.

Zeigler, B. P., and S. D. Chi. 1993. "Model-Based Architecture Concepts for Autonomous Systems Design and Simulation." In *An Introduction to Intelligent and Autonomous Control*, 57-78. Kluwer Academic Publishers.

Zeigler, B. P., and H. Sarjoughian. 2012. *Guide to modeling and simulation of systems of systems (simulation foundations, methods and applications).* London: Spring-Verlag.

Zeigler, B. P., H. Praehofer, and T.G. Kim. 2000. *Theory of Modeling and Simulation.* New York, NY: Academic Press.

Zeigler, B. P. 2004. "Discrete Event Abstraction: An Emerging Paradigm for Modeling Complex Adaptive Systems." In *Perspectives on Adaptation in Natural and Artificial Systems, Essays in Honor of John Holland*. Oxford University Press.

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

AUTHOR BIOGRAPHIES

SAURABH MITTAL is the founder and principal of Dunip Technologies, USA. He is also affiliated with L3 Communications at Air Force Research Lab, WPAFB, OH. He received both a PhD (2007) and MS (2003) in Electrical and Computer engineering from the University of Arizona, Tucson. He can be reached at smittal@duniptech.com

JOSE LUIS RISCO-MARTIN is associate professor at the Department of Computer Architecture and Automation of Universidad Complutense de Madrid. He received both a PhD (2004) and MS (1998) in Physics from Universidad Complutense de Madrid. He can be reached at jlrisco@dacya.ucm.es