OSM: An Evolutionary System of Systems Framework for Modeling and Simulation

1Clinton M. Winfrey 1Benjamin A. Baldwin 1Mary Ann Cummings 2Preetam Ghosh

clinton.winfrey@navy.mil benjamin.baldwin@navy.mil mary.cummings2@navy.mil pghosh@vcu.edu

1Naval Surface Warfare Center, Dahlgren Division, Dahlgren, VA 22448.

2Virginia Commonwealth University, Richmond, VA 23284.

Abstract

 In this paper, we propose a novel Orchestrated Simulation through Modeling (OSM) framework that allows output visualizations and Discrete Event System Specification (DEVS) modeling and simulation (M&S) frames to be developed separately as plug-ins and combined to form a complete system. Independently developed plug-ins can be added and removed as desired to dramatically change the system. This design allows input plug-ins (model, experimental), execution plug-ins (simulator), and output plug-ins to be developed separately and pieced together to form a unique system while allowing development to be compartmentalized. With the OSM framework, an evolutionary system of systems can be intelligently created by a community. Each community member only needs to fully understand the pieces they personally develop. The performance and scalability of our proposed OSM framework is discussed in an evolutionary system of systems domain pointing to its efficiency and usability.

***Keywords:*** DEVS, plug-ins, framework, agent-based modeling, system of systems

# Introduction and Related Work

 A *system of systems* can be defined as a collection of systems that each have their own purposes and will continue to achieve those purposes if removed from the overall system [1]. Evolution occurs when individual systems need to be added, removed, or replaced. Projects often begin as a single system. Over time, a new problem is identified, and a new system is added to meet the needs of the new problem. Old systems are shown to be weak and need to be replaced. Eventually, the system of systems becomes unmanageable and cannot be maintained properly (Figure 1).



Figure 1. Evolution of Unmanaged System of Systems [2]

 The first step to developing an evolutionary system of systems framework is to ensure that it does not have to change as new systems are added. The second step is to ensure that the framework does not require systems to be modified when other systems are added, removed, or replaced [2]. Our proposed Orchestrated Simulation through Modeling (OSM) framework meets both of these criteria.

 An evolutionary system of systems framework has become somewhat of a holy grail for the M&S community. Many organizations have sought after such an approach. The Missile Defense Agency is an excellent example of such an organization. There are many systems involved in shooting an enemy missile out of the sky, and it is difficult to integrate all of the involved models together [3]. The High Level Architecture (HLA) defines a good way to integrate systems [4]. There are drawbacks to this approach, however. Although HLA is an IEEE standard, there is no standardized implementation of the Run-Time Infrastructure (RTI), which is what allows for interactions between these systems. In addition, the experimental setup, simulation algorithm, and feedback to the user are typically limited in HLA implementations.

 Our proposed OSM framework is also ideal for agent-based modeling. Agents are designed to be autonomous, meaning that each agent has its own goals and purposes [5]. Agent-based modeling is a useful way to simplify the complexities of a system. Although one might argue that it is not an ideal way to predict scenarios with precision, it is not necessarily intended to be. Its true strength is in identifying emergent behaviors and characteristics in a system of systems [6] that are independent, even when fairly simple rules are used [7]. Emergence in a system of systems occurs when properties emerge that do not occur in any of the individual systems. Our goal in this paper is to present a framework that can involve subject matter experts, who can help to define models accurately so that very realistic emergent behavior might be realized [8].

 There are various methods for exercising models through simulation algorithms. One theory that has been extensively studied in M&S theory is DEVS, which is essentially a formalism for executing discrete events through time. In many cases, a hybrid system methodology is needed that uses continuous and discrete events [9]. OSM provides an event queue that makes continuous, discrete, and hybrid methods simple to implement. Deciding if and how to use the event queue is the job of the simulator plug-in and the models that schedule events.

 The DEVS method divides simulation into three M&S frames: model, experimental, and simulator [10]. OSM defines a protocol whereby user feedback and DEVS frames can be developed separately as plug-ins and combined to form a complete M&S system. An OSM-based system consists of a collection of input plug-ins (model, experimental), execution plug-ins (simulator), output plug-ins, the OSM executable, and the OSM library. The DEVS formalism is a powerful idea, and OSM’s DEVS-based plug-in approach takes that powerful idea to the next level. This modularity allows for great amounts of reuse and repurposing of code. Plug-ins can be used to give great capability to independent developers [11].

 OSM can be used to address many of the difficulties found in modeling and simulation today. Simultaneous events can use the tie breaking capability provided in OSM’s event queue. The difficulty of collecting statistics is simplified as well – OSM defines the capability for every model to collect statistics for itself, and output plug-ins can be used to interpret the collected data in a meaningful way [12].

# Our Contributions

 As discussed above, OSM fills a gaping void that exists in the current modeling and simulation literature. Where other frameworks fall into the trap of trying to identify and solve many problems for developers, OSM allows developers to solve their own problems and share solutions with the rest of the community. Specifically, the salient features of our proposed OSM framework are as follows.

 *(i) Evolutionary System of Systems Development*: The OSM framework allows a community to develop a truly evolutionary system of systems with minimal community coordination.

 *(ii) Model Plug-ins:* The method in which OSM uses model plug-ins is novel. Model plug-ins in other frameworks serve the purpose of letting developers define an advanced version of a basic model that is already defined within the framework. Where other frameworks try to address a specific domain of models, OSM’s approach is model-agnostic. In addition, the method of creating a model plug-in is very simple, lightweight, and intuitive. Several legacy models have been successfully wrapped into OSM through the model plug-in methodology. Projects are not limited to any specific domain and have included molecule interactions, Pong, ship warfare, satellite movement, missile defense, and more.

 *(iii) Experimental Plug-ins:* The experimental frame defines the experiment. OSM may load multiple experimental plug-ins at startup. The user can select which experiment to run at any time. This enables the reuse of model, simulator, and output plug-ins for different experimental designs.

*(iv) Simulator Plug-ins:* The simulator frame contains the simulation clock and the driving algorithm or simulation engine. OSM may load multiple simulator plug-ins at startup. The user can select (either explicitly or implicitly through the experimental plug-in) which simulator to use. One simulator plug-in may use a discrete events algorithm, while another may use discrete time steps.

*(v) Output Plug-ins:* There are an infinite number of ways to output data (graphics, maps, charts, files, etc). Other frameworks limit output of simulation data by building output mechanisms into the framework. With OSM, any method of output can be implemented. As problems evolve, output to the user must be able to evolve as well.

*(vi) Flexibility:* OSM is flexible to the creativity of the plug-in developers. Developers have implemented several interesting ideas such as composing model plug-ins from component plug-ins, experimental capability to rewind and reset the future, and executing legacy models over a network.

*(vii) Reusability:* As approaching a solution for an M&S problem can be difficult, a starting point is selected based on available time, resources, and funds. OSM’s promotion of reuse simplifies this problem. Instead of starting with a bottom-up approach, developers can reuse the work of others and start climbing at the middle. An M&S solution can be reached by some combination of data, theories, models, and simulations. OSM promotes a solution that touches on all four of these approaches in a meaningful way [13]. Reuse leads to lower development cost, faster development, and fewer errors. With OSM, developers can modularize the development of components and make new plug-ins only as needed. This saves time and money on development, testing, etc. Many times a simple, preexisting experimental or simulator plug-in can be reused for diverse projects. Output and model plug-ins have also proven to be reusable. It is common for large amounts of effort to be duplicated when all that really needs to change is an output or a DEVS frame.

# OSM Operation

 OSM defines a protocol whereby output and DEVS frames can be developed separately as plug-ins and combined to form a complete M&S system. An OSM-based project consists of a collection of input plug-ins (model, experimental), execution plug-ins (simulator), output plug-ins, the OSM executable, and the OSM library (Figure 2). Note: OSM is under patent protection.



Figure 2. Notional Example of OSM project

## OSM Library and Executable

 There are two parts to OSM itself: the library and the executable. The OSM library defines plug-in types, basic interfaces, classes, and functions. The executable loads and organizes the plug-ins so they can interact in a meaningful way. OSM is the nucleus of the system where plug-ins may be interchanged.

## Model Frame Plug-in

 A model plug-in is used to define model instances (agents) that will be exercised through the simulator plug-in’s algorithm. Developing a model plug-in is simple. It allows the user to create model entries in the scenario database. The user can create as many model entries as desired. OSM asks each model plug-in to create agents at the start of simulation. Because the model plug-in defines how its scenario entries are written, it also knows how to read them.

 Many M&S frameworks claim to use model plug-ins, but the methodology described here is unique. The OSM methodology is model agnostic, meaning that anything can be modeled with it. An important part of what makes OSM so powerful is that model interfaces and interactions are all written outside of the framework. As such, an interface that the model uses can be extended, removed, or replaced without affecting the framework itself. OSM will never know what the models are or how they interact. This approach allows light-weight plug-in wrappers to be written for existing model software. This has been demonstrated with C++ models through Java Native Interface (JNI), Java models, and other executable models. Figure 3 shows an example of how a model plug-in works.



Figure 3. OSM Model plug-in development

## Experimental Frame Plug-in

 The experimental plug-in provides a way for the user to set up the experiment. It provides global information to any model that may request it. Such global information may include a randomization seed, start time, or anything else that affects multiple models. The experimental plug-in can also be designed to create and run a design of experiments, allowing the user to probe the sensitivities and interdependencies of the system. It is intended to do the heavy-lifting in high-run-count situations by setting up, running, adjusting parameters, and re-running until the desired parameter space has been covered. OSM provides basic methods whereby the plug-ins may collect data during each run, but it also allows plug-in developers to define their own data collection if desired. The data is then available for use by the output plug-ins described in section 3.5.



Figure 4. Experimental plug-in example

 The simple experimental plug-in GUI (Figure 4) has a seed that can either be randomized at execution or explicitly set by the user for repeatability purposes. Any agent that is dependent on randomization or is sensitive to dates may request such information from this experimental plug-in. The “Max Seconds” field sets a maximum execution time end condition for the simulation. The Time Step is used by any agents that require a discrete time step. This experimental plug-in also allows the user to explicitly select the simulator plug-in, set the number of runs for a batch process, and displays a progress bar.

## Simulator Plug-in

 The simulator plug-in contains the algorithm that exercises the models and manages the simulation clock. This is the main driving engine of the simulation. Examples of simulator algorithms include discrete event queues and various differential equation algorithms.

## Output Plug-in

 Output plug-ins provide feedback to the user. There are an infinite number of ways that data can be output meaningfully. With output plug-ins, data can be output in any way desired. If an output plug-in has a graphical user interface, OSM puts that GUI into a tab that can be popped out, resized, moved to a separate screen, and popped back.



Figure 5. Data viewed in multiple output plug-ins

 Figure 5 shows an output plug-in that uses NASA’s 3-dimensional World Wind mapping library [14]. The smaller window shows a separate plug-in outputting the same data on a 2-dimensional map (OpenMap) [15]. OSM has a time slider that can be used for playback purposes. As the time slider moves back and forth, both maps are updated. Additional mapping plug-ins can be added without changing a single line of code to the models themselves.

 Maps are only one way for the user to visualize data. The project in Figure 5 uses a plug-in to describe the events that occurred during simulation. A chart output plug-in shows metrics that were collected. Another output plug-in shows the state that each model is in at a given time. All of these plug-ins are general enough that other projects may reuse them. Such output plug-ins are an important part in designing an evolutionary system of systems. With OSM, the framework does not need to be changed in order to add new output capability.

## User Interaction

 The user interacts with OSM by defining model instances through the model plug-ins. In addition, the user selects the experimental plug-in and sets up the experiment with it. The experimental plug-in can then give the user the ability to select the simulator plug-in. Finally, the user receives feedback from the simulation of models through output plug-ins.

# Use Case: Predator/Prey

 Predator/Prey is a system of systems example that may evolve as increased complexity is desired. In the beginning, very simple animals may be designed. Eventually, a realistic ecosystem could be created to include trees, caves, water, and weather with animals that could interact with their surroundings in an effort to attack or survive.

## Predator/Prey: Simple System of Systems

 This example demonstrates that a system of systems can be created simply, with only a single model plug-in. This model plug-in has an interface that allows the user to define some basic characteristics of each species (Table 1).

|  |  |  |  |
| --- | --- | --- | --- |
| Species ID | Fox | Plant | Rabbit |
| Color | Red | Green | White |
| Latitude | 37.257 | 37.257 | 37.257 |
| Longitude | -78.551 | -78.551 | -78.551 |
| Variance (meters) | 20 | 5 | 10 |
| Starting Population | 50 | 250 | 200 |
| Speed (meters/sec) | 1 | 0 | 3 +-2 |
| Flee Predators at (meters) | 0 | 0 | 10 +-2 |
| Attack Prey at (meters) | 200 | 200 | 200 |
| Steps Before Death | 1000 | 100 | 60 +-20 |
| Steps Before Reproduction | 1mil | 10 +-6 | 25 +-4 |
| Steps Before Starvation | 25 +-10 | 100 | 20 +-3 |
| Prey | Rabbit |  | Plant |

Table 1. Definitions of three species from one plug-in

 Figure 6 shows that the user has created three species (foxes, rabbits, and plants). The white rabbits are moving towards and eating the green plants. The red foxes are coming from the bottom right to eat the rabbits and have surrounded some rabbits in the center. The only limit to defining more species is computer resources. An entire ecosystem could be defined by this single model plug-in. This is a system of systems, but it is not an evolutionary one. What if a bird needed to be added? That could not be done with the currently defined plug-in, because the third dimension introduced in flight is not currently defined as an attribute. It would be nearly impossible to add all possible characteristics of every species in a single plug-in. If such a plug-in were created, it would be unusable. This plug-in can create a system of systems, but the system of systems cannot evolve.



Figure 6. Multiple species defined by a single plug-in

## Predator/Prey: Evolutionary System of Systems

### Purpose

 This example describes how to make a system of systems project that is capable of evolving. First, interface layers are defined in the form of shared libraries for geographic, environmental, and interspecies interactions. These basic interface layers do not have to be defined perfectly up front, since they can be swapped out in the future. Interface layers should be defined at the basic level with the goal of allowing other agents to interact with them. For example, the water body interface includes a way to get the closest water position, indicate if a position is inside of the water body, and provide the depth at a specific position in the water body. If a new interaction is needed, such as identifying the water’s flow speed, that capability can be added into the primitive library at a later point. An alternative would be for a new interface to extend the primitive interface and define such a characteristic for others to interact with. It is very important that these definitions exist outside of the framework, so they can be updated, removed, or replaced as desired.

### Model Plug-ins

 Next, we demonstrate the addition of four model plug-ins using our OSM framework. The Land Animal plug-in has the same characteristics as that of the simple system of systems plug-in defined in Table 1. Water Drinker is nearly identical to Land Animal, with the execption of a few characteristics that dramatically change the species’ behavior (Table 2).

|  |  |  |
| --- | --- | --- |
| Species ID | Fox | Rabbit |
| Color | Red | White |
| Latitude | 38.438 | 38.438 |
| Longitude | -77.165 | -77.183 |
| Variance (meters) | 10 | 10 +-5 |
| Starting Population | 20 | 50 |
| Speed (meters/sec) | 5 | 10 +-4 |
| Flee Predators at (meters) | 20 | 100 |
| Attack Prey/Water (m) | 3000 | 2k +-1k |
| Steps Before Death | 300 | 900+-20 |
| Steps Before Reproduction | 250 | 250+-20 |
| Steps Before Hunger |  | 30 +-20 |
| Steps Before Starvation | 150 | 200+-20 |
| Steps Before Thirst |  | 30+-20 |
| Steps Before Dehydration |  | 200+-20 |
| Prey | Rabbit | Squash, Carrot |

Table 2. Definition of Land Animal (gray characteristics) and Water Drinker (black)

 Plants are organisms, just like Land Animals and Water Drinkers. However, they behave very differently. They do not move, have prey, or define how to interact with other organisms. They can only be eaten and reproduce. Table 3 describes the two plant types defined by this plug-in.

|  |  |  |
| --- | --- | --- |
| Unique ID | Squash | Carrot |
| Color | Green | Orange |
| Latitude (deg) | 38.44 | 38.451 |
| Longitude (deg) | -77.169 | -77.183 |
| Position Variance (m) | 100 | 100 |
| Starting Population | 3 | 20 |
| Steps Before Reproduction | 125+-25 | 150+-20 |
| Attacks Before Death | 100 | 5 |

Table 3. Definition of Plant

 A pond can answer all of the questions that are required by the water body interface layer. The answers from a pond would differ from those of a stream, because they would be modeled differently. However, the interface is consistent. Two ponds are defined for this simulation (Table 4).

|  |  |  |
| --- | --- | --- |
| Unique ID | Pond1 | Pond2 |
| Latitude (deg) | 38.468 | 38.479 |
| Longitude (deg) | -77.103 | -77.193 |
| Position Variance (m) | 0 | 0 |
| Minimum Radius (m) | 6,000 | 3,000 |
| Maximum Radius (m) | 6,300 | 3,300 |
| Radius Change (m) | 10 | 2 |

Table 4. Definition of Pond

### Additional Plug-ins

 The two output plug-ins used in this project are simply map visualizations. One uses the World Wind library, and the other uses OpenMap. The two simulator plug-ins used in this project are a DEVS algorithm and a time slider algorithm. The DEVS algorithm simply executes discrete events that are defined by the agents. It can be used to quickly execute the simulation for playback purposes. The time slider algorithm executes agents’ events based on OSM’s time slider. This simulator plug-in can allow a human player to control a keyboard animal (defined in a fifth plug-in). The two experimental plug-ins used for this project are a simple one (Figure 4) and a more complex one that allows the user to reset the known future for all animals. Upon reset, the animals may take a new path. This is particularly interesting when the user plays with a keyboard animal to force certain interactions, rewinds what has occurred, resets, and forges a new set of interactions from that point forward.

### Results

 In Figure 7 there are orange dots near the top left that represent carrots and a few green dots to the right of middle that represent squash. The line of red dots to the right are foxes that are pursuing a group of white rabbits. Those rabbits are heading towards the squash and the top-right pond, and another group is heading towards the top-left pond and the carrots. Sometimes, the rabbits eat all of the carrots, run out, and migrate towards the squash. At that point, the rabbit population from the top is large enough that the foxes never run out of food. In other cases, the rabbits all head towards the plants and are eaten to extinction.



Figure 7. Models defined in project plug-ins, interacting as an evolutionary system of systems

 Plant (carrot, squash), Land Animal (fox), and Water Drinker (rabbit) plug-ins all define species, but they behave very differently. Plants never move and can be eaten without dying. Land Animals eat constantly. Water Drinkers eat when hungry and drink when thirsty. The plug-in defines the true behavior of a model, and interface layers define how the models interact (Figure 8).



Figure 8. Interface layers (black) and model plug-ins (blue)

 An organism has no idea what a water body is, but an organism model can be defined to interact with a body of water by referencing the proper interface layer. In OSM it is easy for agents to share information with each other, even though they may be defined by different plug-ins. Species created by the Water Drinker plug-in can seamlessly interact with bodies of water created by the Pond plug-in. Although this is a simple example, there is no reason that it could not grow to be as complex and realistic as desired. New developers only have to understand the interface layers to define agents that interact with preexisting agents.

### Scalability



Figure 9. 10 plants, 1 water source, 30 steps per execution

 The OSM framework does not limit developers any more than the Java programming language itself. Any constraints are the result of the quality of the plug-ins and the system hardware. As with any agent-based project, more agents equate to a longer execution time. Figure 9 shows multiple executions of rabbits taking 30 steps in the simulation. For each run, there are 10 invincible plants to feed the rabbits and a single pond to water the rabbits. The rabbits cannot die or reproduce. The results show that OSM itself does not change the scalability of the models. There can be as many agents as the machine resources can support. Of course, thousands of agents interacting with thousands of complementary agents can be exponentially complex unless optimization steps are taken.

# Comparing OSM To NetLogo

 NetLogo is defined as a multi-agent programmable modeling environment [16]. It has been used to develop many different kinds of agent-based M&S projects. NetLogo is written in Java, but it defines a powerful M&S-friendly language for developers to use. For comparison purposes, we will discuss the Wolf Sheep Predation [17] and the Rabbits Grass Weeds [18] NetLogo projects. We will compare them to the evolutionary OSM predator/prey project described previously.

 The Wolf Sheep Predation project is written as a single file. The experimental frame consists of two buttons: “setup” and “go”. The simulation frame is started with the “go” button. As in all NetLogo projects, there is a speed changer. The model frame allows the user to change the grass, sheep, and wolf settings. Grass may be toggled on/off, and there is a growth/regrowth time adjuster. Sheep and Wolf may be defined with three adjusters: 1) initial number, 2) gained-from-food, and 3) reproduce. Outputs for this project include a World view and a populations plot. The Rabbits Grass Weeds project has a lot of code that is exactly the same as that found in Wolf Sheep Predation, but they are shared through copy/paste. Maintaining both projects requires making changes to both with minimal code reusability. The two projects are not designed to interact in any way.

 These species are simple enough that they could easily be defined with one OSM model plug-in. However, as discussed previously, an evolutionary system of systems would be easier to design with many types of model plug-ins in OSM. Creating an evolutionary system of systems with NetLogo would be impractical. The code would be unmanageable, as has always been the case with an evolving system of systems. As previously discussed, OSM was designed to facilitate evolutionary systems of systems modeling. The modularity and extensibility offered by OSM can be used to avoid the many pitfalls that occur in system of systems modeling today.

 The experimental frame for Wolf Sheep Predation and Rabbits Grass Weeds is built into each project’s single file of source code. With OSM, many different experimental frame plug-ins can be written, dropped in, and used to change the goals of the experiment or the way the experiment is run. Switching between them is as simple as using a dropdown box or command line parameter.

 The simulation algorithm of a NetLogo project is built into the modeling environment and is strictly played one step at a time. It is not possible to pause, rewind, or go forward to view different states of a simulation. Only the current state is visualized during execution. In OSM, a simulation plug-in can be written to execute the entire simulation up front and viewed in playback or, if desired, a simulation plug-in can provide an algorithm that executes through time.

 Output viewing capabilities are built into the NetLogo tool. It appears that they try to identify all of the meaningful ways for developers to look at data. This is an impossible task, although they have done a good job of providing many useful capabilities. OSM, on the other hand, defines no ways at all to look at data. Nothing is built in, but anything can be added through an output plug-in. These output plug-ins can be developed by one and shared by many, or they can be project-specific. In addition, interface layers can be developed and shared as libraries to provide common interfaces and reusable elements for the community. Thus, our proposed approach is very customizable.

# Conclusion

 In this paper, we have presented a novel M&S framework called OSM that can efficiently characterize the evolution of a system of systems. OSM allows capabilities where many users can develop individual systems, tie them together, replace old modules, build new ones, and let the system of systems naturally evolve with very little communication throughout the community. Where other frameworks try to solve many problems, OSM only has one purpose: to empower individual developers to solve the problems themselves and seamlessly share their solutions with each other. We also presented some case studies to demonstrate the versatility of OSM. Specifically, our case study reveals how an evolutionary system of systems can be developed with output and DEVS plug-ins to form a complete, modular system.

# Acknowledgments

 The authors of this paper wish to acknowledge Dr. Mark Anderson, Dr. Thomas Holland, Dr. Christian Wahlquist, and Sara Wallace for reviewing this paper and providing feedback.

# References

[1] Eberhardt Rechtin and Mark W. Maier. 1997. The Art of Systems Architecting. CRC Press, Inc., Boca Raton, FL, USA.

[2] S. Selberg and M. A. Austin. 2012. Toward an evolutionary system of systems architecture. Technical report, Institute for Systems Research, University of Maryland, USA.

[3] Robert K. Garrett, Jr., Steve Anderson, Neil T. Baron, and James D. Moreland, Jr.. 2011. Managing the interstitials, a System of Systems framework suited for the Ballistic Missile Defense System. Syst. Eng. 14, 1 (March 2011), 87-109. DOI=10.1002/sys.20173 http://dx.doi.org/10.1002/sys.20173

[4] Byeong Soo Kim, Chang Beom Choi, and Tag Gon Kim. 2013. Multifaceted modeling and simulation framework for system of systems using HLA/RTI. In Proceedings of the 16th Communications & Networking Symposium (CNS '13). Society for Computer Simulation International, San Diego, CA, USA, , Article 4 , 7 pages.

[5] O. Thomas Holland. 2007. Taxonomy for the modeling and simulation of emergent behavior systems. In Proceedings of the 2007 spring simulation multiconference - Volume 2 (SpringSim '07), Vol. 2. Society for Computer Simulation International, San Diego, CA, USA, 28-35.

[6] Kishoj Bajracharya and Raphael Duboz. 2013. Comparison of three agent-based platforms on the basis of a simple epidemiological model (WIP). In Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (DEVS 13). Society for Computer Simulation International, San Diego, CA, USA, , Article 7 , 6 pages.

[7] K. A. Hawick, C. J. Scogings, and H. A. James. 2004. Defensive spiral emergence in a predator-prey model. In Proc. Complexity 2004. 14 pages.

[8] Christopher J. Lynch, Saikou Y. Diallo, and Andreas Tolk. 2013. Representing the ballistic missile defense system using agent-based modeling. In Proceedings of the Military Modeling & Simulation Symposium (MMS '13). Society for Computer Simulation International, San Diego, CA, USA, , Article 3 , 8 pages.

[9] Hesham Saadawi and Gabriel Wainer. 2013. Hybrid systems modeling and verification with DEVS (WIP). In Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (DEVS 13). Society for Computer Simulation International, San Diego, CA, USA, , Article 25 , 6 pages.

[10] Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. Theory of Modeling and Simulation (2nd ed.). San Diego, CA, USA: Academic Press.

[11] Fernando J. Barros. 2013. On the representation of product lines using pluggable software units: results from an exploratory study. In Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (DEVS 13). Society for Computer Simulation International, San Diego, CA, USA, , Article 1 , 8 pages.

 [12] Rhys Goldstein, Simon Breslav, and Azam Khan. 2013. Informal DEVS conventions motivated by practical considerations (WIP). In Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (DEVS 13). Society for Computer Simulation International, San Diego, CA, USA, , Article 10 , 6 pages.

[13] Jose J. Padilla, Andreas Tolk, and Saikou Y. Diallo. 2013. M&S methodological challenges. In Proceedings of the Emerging M&S Applications in Industry & Academia / Modeling and Humanities Symposium (EAIA and MatH '13), Charles Turnitsa, Paul A. Youngman, Ted Carmichael, and Mirsad Hadzikadic (Eds.). Society for Computer Simulation International, San Diego, CA, USA, , Article 10 , 9 pages.

[14] Hogan, P. 2004-2013. World Wind. http://worldwind.arc.nasa.gov/java. Ames Research Center, National Aeronautics and Space Administration (NASA), Mountain View, CA

[15] Dietrick, D. 2013. OpenMap. https://code.google.com/p/openmap. BBN Technologies, Cambridge, MA.

[16] Wilensky, U. 1999-2013. NetLogo. http://ccl.northwestern.edu/netlogo. Center for Connected Learning and Computer-Based Modeling, Northwestern Institute on Complex Systems, Northwestern University, Evanston, IL.

[17] Wilensky, U. 1997. NetlLogo Wolf Sheep Predation model.\_http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation. Center for Connected Learning and Computer-Based Modeling, Northwestern Institute on Complex Systems, Northwestern University, Evanston, IL.

[18] Wilensky, U. 2001. NetLogo Rabbits Grass Weeds model.\_http://ccl.northwestern.edu/netlogo/models/RabbitsGrassWeeds. Center for Connected Learning and Computer-Based Modeling, Northwestern Institute on Complex Systems, Northwestern University, Evanston, IL.