Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

An XML Based System of Systems Agent-in-the-Loop Simulation Framework using Discrete Event Simulation

Matthew Hosking
Department of Computer Engineering
Rochester Institute of Technology
Rochester, NY, USA

Dr. Ferat Sahin
Department of Electrical Engineering
Rochester Institute of Technology
Rochester, NY, USA

***Abstract*—This paper extends an XML based system of systems simulation framework using DEVS to support hardware-in-the-loop simulations for SoS. A system of systems approach enables the simulation and analysis of multiple complex, independent, and cooperative systems by concentrating on the data transferred among systems. This paper wraps information exchanged among heterogeneous systems in XML to enable receiving systems to correctly parse and interpret the information. A Groundscout robot is deployed as a real agent working cooperatively with virtual agents in a robotic swarm. The DEVS activity concept facilitates communication between the real system and virtual systems in the SoS. A robust threat detection example is provided. Initial performance metrics of the SoS are briefly discussed.**

***Index Terms*—Discrete Event Simulation, DEVS, XML, System of Systems, Swarm Behavior, Groundscout robots, Hardware-in-the-loop simulation**

I. Introduction

The concept of Systems of Systems (SoS) is essential to more effectively implement and analyze large, complex, independent, and heterogeneous systems [1], [2]. SoS are comprised of systems which themselves are independent and complex systems that interact among each other to achieve a common goal [1]. For example, a Boeing 747 airplane is a system *of* an SoS, but an airport *is* an SoS. The SoS concept is still in the developing stages and several formal definitions are available [3], [4]. For this work we considered the following definition: *SoS are large-scale concurrent and distributed systems that are comprised of complex systems*. This is an *Information Systems* view as it emphasizes the interoperability and integration properties of an SoS [1].

Considering a military operations example, it is obvious that many different ground, air, sea, and space units contribute data to the SoS. Information may be simple sensor data or it may be complex data from an aircraft carrier. Various command centers attempt to aggregate data and inform their subsystems to accomplish the goals of the military. Each of the systems could be developed at different times and with different hardware/software. This can create a barrier to data aggregation and meeting goals of the SoS if the systems cannot interact and communicate. One solution to achieve interoperability is to standardize the communication medium among the systems. Two methods are presented in [2]:

- Create a *software model* which each component in the SoS talks to the module embedded in itself
- Describe the data in a *common language* which each component in the SoS can understand

There is the potential for large overhead in generating software models. Each new member of the SoS would require the re-generation of software models and this approach also assumes, incorrectly, that the complete state-space model is available or practical to describe. In light of the difficulties of software models, this work promotes the use of XML to standardize the communication among the systems in the SoS. A data-driven approach avoids the risk of potentially large overhead and more readily supports legacy components which may not have a model readily available [5]. The data exchanged among the members of the SoS is also captured and processed later to determine the degree of success to which the SoS completes its goal.

The Levels of Conceptual Interoperability Model (LCIM) [6] distinguishes varying degrees of interoperability from technical levels to a conceptual level enabling automated re-use of simulations over a network. XML is also recommended for data exchange as part of an open standard supporting conceptual interoperability in [7]. In a multi-agent system the agents operate autonomously but it is very important they cooperate with other agents to take better actions for the overall goal of the SoS. Interoperability in this example requires each system to exchange data according to an XML standard which is commonly understood in the SoS.

The integration property implies that systems can communicate and interact with the SoS regardless of their hardware and software characteristics, operating systems, and internal data format. As systems enter and leave the SoS their data will be made available and aggregated with existing systems' data.

In [1], an SoS simulation framework based on the Extensible Markup Language (XML) is developed in order to wrap data originating from different subsystems in a common way to address the interoperability and integration requirements of an SoS.

References [8] and [9] employ the basic framework in a

conventional simulation of swarm agents. The new framework bridges simulation and implementation through the use of a system of systems approach to hardware-in-the-loop simulation using discrete events [10]. This paper provides further insight into the framework and presents initial results on a semi-automated method of measuring the performance of swarm behavior.

II. XML SoS AIL Simulation Framework

The desire to have a seamless migration from initial development simulations to deployed systems in the field is referred to as 'model continuity'. A four step process for general model continuity is found in [11]: conventional simulation, real-time simulation, HIL simulation, real implementation and execution. The HIL simulation step allows part of the final system to be implemented and debugged before more resources are directed towards a flawed product. In [12] a rig is built to perform HIL testing of aerial vehicles' sensors and flight algorithms without the need to fly the UAV. A simulation based virtual environment to study cooperative robotics is discussed in [13] and [14] specifically presents "robot-in-the-loop" simulations. An integrated test bed for robots utilizing an overhead camera system for monitoring the system is presented in [15].

A. DEVS

The interactions between the independent systems within an SoS are asynchronous in nature and can be effectively represented as discrete event models [1]. Discrete Event System Specification (DEVS) [16] is a formalism which provides a means of specifying the components in a discrete event simulation.

As stated formally using set theory notation, an atomic model in DEVS is a structure $M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$ [16] where

X: is the set of input values
S: is a set of states
Y: is the set of output values
$\delta_{int} : S \rightarrow S$ is the internal transition function
$\delta_{ext} : Q \times X^b \rightarrow S$ is the external transition function,
$\delta_{ext} : Q \times X^b \rightarrow S$
where $Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is total state set
and e is the *time elapsed* since last transition
X^b denotes the collection of bags over X
(bags are sets in which some elements may occur more than once)
$\delta_{con} : Q \times X^b \rightarrow S$ is the confluent transition function
$\lambda : S \rightarrow Y^b$ is the output function
$ta : S \rightarrow R^+_{0,\infty}$is the *time advance* function

Basic models, called *atomic* models, can be connected together to form larger, more complex models called *coupled* models. A SoS hierarchy of systems is easily organized and simulated in this context. *Couplings* among atomic models are the connections which specify the relationships between one DEVS model's output port and another model's input port. An output value from one model is transferred in a `message` object to be received as an input value for a second model if a coupling exists between them. DEVS coupling is a closed operation as each coupled model can be represented as an atomic model [16]; thus, the system-subsystem hierarchical organizational view in a system of systems approach is intuitively supported by the DEVS formalism.

DEVSJAVA [17] is used as the simulator for the DEVS models. This Java implementation of the DEVS formalism supports different types of simulations in the model continuity process; a real time centralized simulator is chosen to execute the virtual systems in this work.

B. Virtual Environment

The environment contains a virtual representation of objects in the system: active systems and passive obstacles in the simulation. To facilitate the virtual agents' interaction and cooperation with real agent, the real agent has a corresponding virtual representation in the environment. If this virtual model is kept up to date with the information received from the real agent in the loop, the two worlds' current state will remain synchronized.

C. Interface with Real World

In previous work, the robots used in the simulation locally execute a lightweight version of the DEVSJAVA simulator and interact with the virtual robots via the distributed real-time simulator framework in the software [18].

The Groundscout robot's 8051 based microprocessor cannot locally execute a DEVSJAVA simulator. The communication with the DEVSJAVA simulation is enabled by DEVS activities. The synchronization of the current state between the two worlds is provided by the representation of the real robot within the virtual environment. The communication and control layers work together to achieve the synchronization of the real robot with the virtual systems.

The control layer in a real robot's virtual representation is a synchronization model used to interface with the virtual environment model as is shown in Fig. 1. A simulated robot's control layer contains the algorithm or behavioral control to define its actions in the SoS as well as the abstract activity used to interface with the environment. The similar design between a real and virtual agent reflects model continuity principles and allows the real robot to be hidden from view of the other virtual systems.

The communication layer's activity listens for updates to the incoming serial packet queue using the Java *Observer-Observable* API. The activity is an `Observer` to the `Observable` serial port packet buffer. The activity extracts the XML messages from the packets and makes them available to the communication layer via the DEVS external transition function. The data packets arrive as well-formed packets of bytes containing header information and a data envelope with XML encapsulated data. The communication layer forwards the message to the control layer and/or broadcasts the message to other virtual systems. The control synchronization layer then

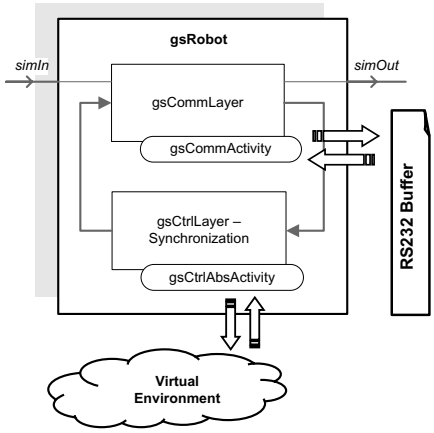

Fig. 1. Communication and Synchronization of Robot Model

```
<!–Created 12/01/2008 Author @ Matt Hosking–>
<q>
    <y>
        <i>ID of the system</i>
        <s>
            <i>ID of the first sensor</i>
            <d>sensor data</d>
        </s>
        <s>
            <i>ID of the second sensor</i>
            <d>sensor data</d>
        </s>
    </y>
    <y>
        <i>ID of the 2nd system</i>
        <s>
            <i>ID of the sensor</i>
            <d>sensor data</d>
        </s>
    </y>
</q>
```

Fig. 2. An XML based SoS architecture implementation

makes a request to the virtual environment when a new location or sensor data from the real robot is contained in the message. The response from the virtual environment is given to the control layer and it then broadcasts the feedback to the virtual systems via the communication layer's output port. Thus, the real robot is transparent and requires no special handling on the part of the virtual systems.

The Groundscout robot tethers to the computer using an RS232 serial connection, but the methodology of connecting and synchronizing a real system with the virtual world is easily applicable to any communication medium from TCP/IP to custom systems.

D. XML Messages

The XML architecture shown in Fig. 2 arises from the necessity to fit data within the storage available in the communication layer of the Groundscout. The IDs of the system and the sensors are single bytes but these are transformed into more user readable `String` types in Java by the communication layer's `activity`. This simple transformation between two ways of representing the data demonstrates XML's flexibility using XLST to help understand the meaning of the XML tags.

Within the simulator, XML string messages are wrapped as `XmlStrEntity` objects to inherit the properties of an `entity` to be sent as `messages` between different components. This class also provides basic parsing and encoding methods for the simple XML architecture in use.

III. Modeling

Each component in the SoS is described as a DEVS model. In general, an atomic model in DEVS would correspond to a single system in the SoS, and coupled models are created to show the hierarchical relationship among the systems and their paths of communication. The designer, however, can be as detailed as desired in describing the model of a single system. Each robot in the system is created as a coupled model to reflect the actual hardware. This provides a more thorough verification of the agent and follows model continuity principles.

A. Command Center

The command center serves as an interface between the multiple stationary sensors and the robot team. In this example, the command center searches for the presence of a threat in the incoming data from each sensor. When a threat is detected, the command center broadcasts a target location to mobilize the robotic swarm. If the command center is tracking a threat which is no longer detected by any radar station, the robotic swarm is also notified.

Fig. 3 shows the `serialPortThread` contained in the command center. This is the thread responsible for handling operating system level calls to the serial port on the workstation containing the virtual systems. Incoming and outgoing packets are each stored in a buffer which is observed by the virtual model's communication activity. The flow of data operates independently of the command center control and does not feed into the command center but only serves as an intuitive place to locate this part of the framework.

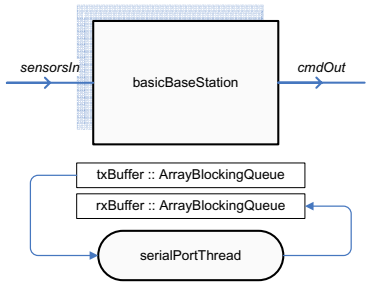

Fig. 3. DEVS model of command center

B. Radar Stations (Sensors)

Each radar station can detect a threat within its range. When a threat is detected, or when a threat leaves its sensor area, this is communicated to the command center. The exact threat location cannot be determined by the sensor itself, so only the location of the sensor is sent to the command center.

C. Mobile Threat

A threat is any undesirable agent: a chemical spill in a factory, a fire in a national forest, or in this example, an enemy tank within a friendly zone. The threat moves around randomly and is never disabled. This enables a more thorough investigation of the swarm operating over time.

D. Mobile Swarm Agent (Groundscout)

Mobile swarm agents receive target coordinates from the command center and navigate autonomously to the destination. They communicate and cooperate with one another to more efficiently use their resources. When no threat is present in the area, the swarm agents return to their home positions.

The Groundscout robot has separate control and communication layers [19] which are modeled in the *communication* and *control* layers. In this way the modeling reflects the architecture of the robots closer than our previous modeling efforts [9].

1) Communication Layer: The communication layer is responsible for sending and receiving the XML data to the other systems. There is an inport, `rxSimMsg`, and an outport, `txSimMsg` to facilitate this external communication as shown in Fig. 4. The communication layer also must connect to the control layer in order to forward data to it and transmit any requested information to other systems. `ctrlLayIn` and `ctrlLayOut` provide this path.

To facilitate communication with the real Groundscout robot, the DEVS model includes `gsCommActivity`. This activity runs as a Java `Thread` and has access to the serial port's transmit and receive buffers for data packets. The activity retrieves packets from the receive buffer if it belongs to the model and places outbound data packets in the transmit buffer.

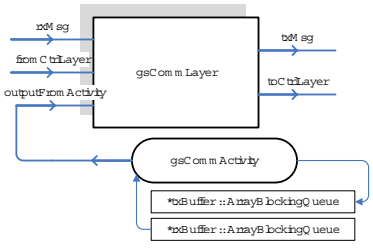

Fig. 4. DEVS model of `gsCommLayer`

2) Control Layer: Behavior of the swarm agent is determined by the algorithm used in the control layer of the model. The control layer uses the communication layer to send and receive packets from other systems and uses `gsCtrlAbsActivity` as an interface to the virtual environment. Distance sensor readings and position updates are sent from the environment when the activity requests a move to be completed. Fig. 5 shows the general DEVS model for the control layer of the coupled robot model.

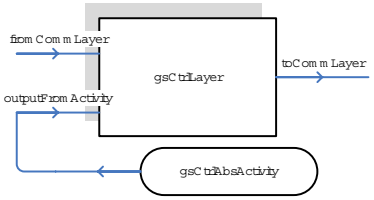

Fig. 5. DEVS model of `gsCtrlLayer`

The control layer implements basic swarm behavior if it is part of a virtual agent model, but implements synchronization logic if it is part of the model representing the real Groundscout. This keeps the state of the robot updated between the real and virtual worlds. It is also possible for distributed control of the real robot: some control is deployed on the robot locally while other, perhaps more advanced algorithms or goals, would be deployed in the DEVS control layer model along with the synchronization model.

IV. Evaluating Swarm Behavior

In the context of general swarm behavior, it may be difficult to have a concrete set of metrics by which to analyze the system. We distinguish between the performance of an individual robot and the emergence of swarm behavior by the robot when cooperating with other systems.

The performance of a robot is closely related to the technical specifications claimed by the unit. Position, heading, speed of travel, rate of communication, and accuracy of sensor readings can be measured very objectively. These factors may or may not affect the emergence of swarm behavior depending on the desired goal of the SoS swarm.

Convoy speed, number of adjustments, formation coherence, scalability, and sensitivity are given as performance metrics for the robot following convoy example in [20]. In the threat detection example of this paper, there is no predefined formation to expect or number of adjustments required to maintain this formation. Additionally, the swarm behavior will not cease when a goal has been met because the SoS will continue to evolve and meet other goals. The formal analysis of such concepts is still a developing area [21], [22].

The use of an overhead camera system to provide global state data for analysis is not practical for real applications of swarm robotics and may not even be an option if we are considering, for example, UAVs or an entire military division. Other means of analysis, such as tracking and aggregating interactions and communications from the other systems in the SoS, must be employed. In perhaps the most subjective example, human 'systems' provide the feedback of whether or not the desired behavior was exhibited. In additional to this metric of pass/fail given a certain experimental setup, data from other systems could be used to verify the agent in the simulation loop is responding and taking cooperative actions

according to the swarm behavior and for the good of the SoS' goal of detecting a threat.

The desired swarm behavior of the mobile systems in the threat detection SoS is to converge on a given target location and yield to a teammate who is closer to the threat target. The goal of the robot team is to investigate a threat before it leaves the sensor network area.

V. Robust Threat Detection Example

In this example, a network of sensors detects an enemy threat and report their readings to a central command center which dispatches a team of mobile agents to investigate the area. The scenario follows that of [8] and [9]. The helicopters cooperate using basic swarm behavior and this allows the helicopter closest to the supposed tank location to take the lead investigating while the other helicopters in the team hold their positions a distance from the lead helicopter.

(a) Radar (b) Helicopter (c) Enemy Tank

Fig. 6. Graphics used in simulation

The results of the threat detection simulation are presented in this section to illustrate the behaviors of the components in the SoS and the successful communication using XML data messages. Fig. 6 depicts the graphics used in the simulation example. A circle surrounding a radar station gives notice of the range at which it can detect the enemy tank. Seven radar stations form a sensor network area illustrated by the adjacent circles in Fig. 7. The radar stations are named left to right and from top to bottom so that the first station, `sr81`, is on the top left, `sr84` is in the center of the network, and `sr87` is the lower right station. Fig. 7 also shows three scout helicopters stationed around the radar network. The darker green helicopter, `gs02`, stationed in the bottom left corner of the plot is deployed as a Groundscout robot in the lab environment. The other two lighter, yellow helicopters are virtual systems in the simulation.

The simulation begins in Fig. 7; as the simulation continues a radar station detects a threat and notifies the command center. The command center transmits the following XML message to the scout helicopters:

```
<y>
   <i>bs09</i>
   <s>
      <i>+</i><d>509,384</d>
   </s>
</y>
```

The scout helicopters receive this message and interpret the parsed coordinate data as the destination location to investigate. The swarm agents communicate with one another using XML messages containing their current locations:

```
<y>
   <i>gs02</i>
   <s>
      <i>$</i><d>384,384</d>
   </s>
</y>
```

Fig. 8 shows `gs02`, the Groundscout robot deployed in the lab, over `sr85` with another virtual agent holding in close proximity.

When the enemy tank is no longer detected and the lead helicopter has investigated the last reported location of the threat, all swarm agents return to their initial locations and wait for an XML message from the base station.

It is observed through the AIL simulation that the Groundscout robot does exhibit the desired swarm behavior. The robot travels towards the threat location and yields to other team members or emerges as the leader. Motion towards a home location is executed while the threat is not visible to the sensor network

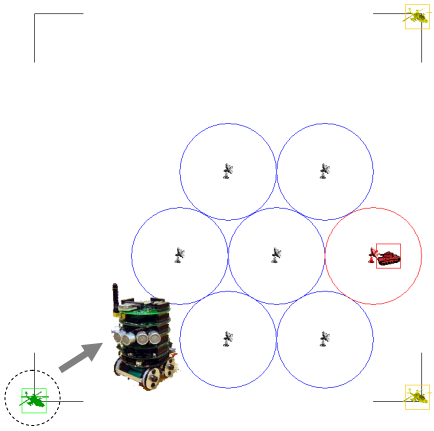

Fig. 7. Initial positions of agents in the SoS [10]

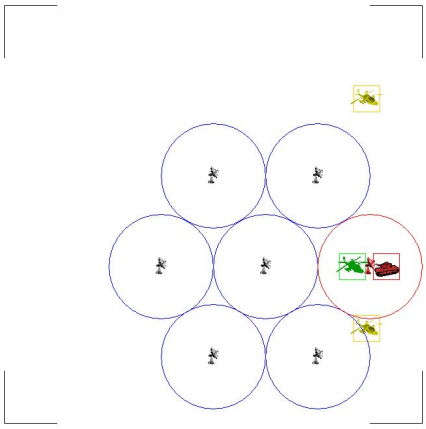

Fig. 8. Scout helicopters intercept enemy tank

The emergent behavior of the robot team achieves a goal bigger than a single robot can handle through the successful

exchange of information within the example SoS. The goal of the example SoS is to investigate a threat; it is expected that as more robots are deployed to the area the more likely the threat will be investigated. Any robot, real or virtual, arriving at the threat's location before it leaves the area is counted as a success. The success rate increases non-linearly but begins to saturate as it nears 100% in Fig. 9.

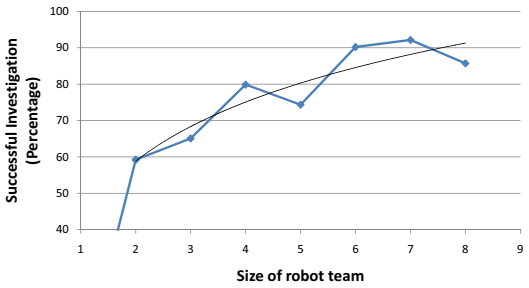

Fig. 9. Percentage of successful threat investigations vs. swarm size

VI. Conclusion

It is important not only to be able to efficiently model and simulate systems of systems, but to integrate these simulations with the systems deployed in the field. It is desirable to maintain the same modeling from simulation to implementation and verify the interoperability of the system within the SoS before a full scale implementation. An XML based system of systems agent-in-the-loop simulation framework accomplishes this task and provides an important step in the model continuity process. Interoperability is achieved by wrapping the data exchanged among the systems with commonly understood XML tags. DEVS activities provide a link between the simulation world and the physical world. The framework presented in this paper has been verified through a successful simulation of a robust threat detection using a Groundscout robot working cooperatively with virtual swarm agents. The robot team met the desired system goal of investigating a threat. As the number of robots in the example SoS increased, so did the success rate. The collection and analysis of further evidence will facilitate more concrete methods of analyzing the behavior in the future.

More data will be captured in varying simulation runs and analyzed for more performance metrics. These may include the effects of communication loss on swarm performance and the robustness of the swarm algorithm independent of meeting the higher goal of the SoS. Additionally, the XML structure will continue to be developed into a proposed standard for systems of systems based approaches to analyze and implement large, complex, and distributed systems.

References

[1] F. Sahin, M. Jamshidi, and P. Sridhar, "A discrete event xml based simulation framework for system of systems architectures," in *Proc. IEEE International Conference on System of Systems Engineering SoSE '07*, 2007.

[2] F. Sahin, P. Sridhar, B. Horan, V. Raghavan, and M. Jamshidi, "System of systems approach to threat detection and integration of heterogeneous independently operable systems," in *Proc. IEEE International Conference on Systems, Man and Cybernetics*, 2007.

[3] M. Jamshidi, Ed., *System of Systems - Innovations for the 21st Century*. Wiley & Sons, 2008.

[4] ——, *System of Systems Engineering*. CRC Press, 2008.

[5] S. Mittal, B. P. Zeigler, J. L. R. Martin, F. Sahin, and M. Jamshidi, *Modeling and Simulation for System of Systems Engineering*. Wiley & Sons, 2009, ch. 5 in System of Systems Engineering - Innovations for the 21st Century, pp. 101–149.

[6] A. Tolk and J. A. Muguira, "The levels of conceptual interoperability model (lcim)," in *Proc. Fall Simulation Interoperability Workshop*, September 2003. [Online]. Available: http://www.sei.cmu.edu/isis/pdfs/tolk.pdf

[7] A. Tolk, "Composable mission spaces and m&s repositories - applicability of open standards," in *Spring 2004 Simulation Interoperability Workshop (SIW)*, 2004.

[8] C. Parisi, F. Sahin, and M. Jamshidi, "A discrete event xml based system of systems simulation for robust threat detection and integration," in *Proc. of IEEE/SMC International Conference on System of Systems Engineering*, June 2008.

[9] M. Hosking and F. Sahin, "An xml based system of systems discrete event simulation communications framework," in *SCS Spring Simulation Multiconference*, 2009.

[10] ——, "A discrete xml based system of systems hardware-in-the-loop simulation for robust threat detection," in *IEEE Fourth International Conference on System of Systems Engineering*, 2009.

[11] Y. K. Cho, B. P. Zeigler, and H. S. Sarjoughian, "Design and implementation of distributed real-time devs/corba," in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, 7–10 Oct. 2001, pp. 3081–3086.

[12] V. Narli and P. Y. Oh, "Hardware-in-the-loop test rig to capture aerial robot and sensor suite performance metrics," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, P. Y. Oh, Ed., 2006, pp. 3521–3526.

[13] X. Hu and B. Zeigler, "A simulation-based virtual environment to study cooperative robotic systems," *Integrated Computer-Aided Engineering (ICAE)*, vol. 12, no. 4, pp. 353–367, November 2005.

[14] X. Hu, "Applying robot-in-the-loop-simulation to mobile robot systems," in *Proc. International Conference on Advanced Robotics ICAR '05*, 2005, pp. 506–513.

[15] H. Azarnoush, B. Horan, P. Sridhar, A. Madni, and M. Jamshidi, "Towards optimization of a real-world robotic-sensor system of systems," in *Proc. World Automation Congress WAC '06*, B. Horan, Ed., 2006, pp. 1–8.

[16] B. Zeigler, "Devs today: recent advances in discrete event-based information technology," in *Proc. 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems MASCOTS 2003*, 2003, pp. 148–161.

[17] B. Zeigler and H. Sarjoughian, *Introduction to DEVS Modeling and Simulation with JAVA*. www.acims.arizona.edu, 2001.

[18] X. Hu and B. Zeigler, "Model continuity in the design of dynamic distributed real-time systems," *IEEE Transactions on Systems, Man and Cybernetics*, 2005.

[19] F. Sahin, "Groundscouts: Architecture for a modular micro robotic platform for swarm intelligence and cooperative robotics," in *Proc. IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[20] X. Hu and B. Zeigler, "Measuring cooperative robotic systems using simulation-based virtual environment," in *Performance Metrics for Intelligent Systems Workshop*, August 2004.

[21] A. F. T. Winfield, C. J. Harper, and J. Nembrini, "Towards dependable swarms and a new discipline of swarm engineering," in *Simulation of Adaptive Behaviour, workshop on Swarm Robotics SAB'04*, 2005.

[22] A. F. Winfield and J. Nembrini, "Safety in numbers: fault-tolerance in robot swarms," *Int. Journal Modelling, Identification and Control*, vol. 1, no. 1, 2006.