2004 IEEE International Conference on Systems, Man and Cybernetics

Discrete Event Modeling and Simulation: V-Lab®*

Application to Wireless Sensor Networks

Prasanna Sridhar
Department of Electrical and Computer Engineering
The University of New Mexico
Albuquerque, NM, U.S.A.
prasanna@ece.unm.edu

Mo Jamshidi
Department of Electrical and Computer Engineering
The University of New Mexico
Albuquerque, NM, U.S.A.
jamshidi@ece.unm.edu

Abstract - *The need for modeling and simulation (M&S) is seen in many diverse applications such as multi-agent systems, robotics, control systems, software engineering, complex adaptive systems, and homeland security. With the emerging applications of multi-agent systems, there is always a need for simulation to verify the results before the actual implementation. Multi-agent simulation provides a test bed for several soft computing algorithms like fuzzy logic, neural networks (NN), probabilistic reasoning (Stochastic Learning Automata, Reinforcement learning), and evolutionary algorithms (Genetic Algorithms). Fusion of soft computing methodology with existing simulation tools yields several advantages in simulating multi-agent systems. Such a fusion provides a novel and systematic way of handling time-dependent parameters in the simulation without altering the essential functionality and problem-solving capabilities of soft computing elements. The fusion here is the extension of the capabilities of simulation tools with intelligent tools from soft computing. This paper proposes a methodology for combining the agent-based architecture, discrete event system and the soft-computing methods in the simulation of multi-agent systems and defines a framework called Virtual Laboratory (V-Lab®) for realizing such multi-agent system simulations. Detailed experimental results obtained from simulation of robotics agents and wireless sensor network is also discussed.*

1 Introduction

1.1 Motivation

Several specialized multi-agent simulation toolkits are available; each of which performs narrow tasks very well. Typically, the development of these specialized simulation toolkits depends on the needs and available resources. Although a general toolkit for simulation of multi-agent systems cannot compete with a specialized toolkit, the general toolkit arguably provides several tools that can be used and reused in several different multi-agent applications. Such a general purpose simulation driven design process requires models to be flexible, scalable and reusable. Furthermore, the whole design process of the simulation framework should greatly benefit from the modular model architecture, that is, each system model is a composition of sub-models that interact in meaningful fashion. The multiplicity of modeling approaches to be integrated as well as the different requirements of potential user groups discourages the development of a unified modeling and simulation system. Instead of this, we can rather propose a modeling and simulation framework, which allows the development of simulation models on different levels, according to user specific abilities and goals. Experimenting with the models requires the iterative and multiple applications of model initializations (e.g. empirical data input), simulation runs, model analysis methods, and presentation of results. Combining these time consuming tasks in a comfortable and transparent way is an essential requirement of a simulation system.

1.2 Discrete Event Simulation

In the discrete time simulation, the system changes state only at certain time steps. In most of the cases, these time slices are the same, which causes the system to change state at a regulated amount of time. In the discrete event simulation, the state change is governed by the occurrence of event or message to the system.
Discrete Event System Specification (DEVS) [1] is a formalism, which provides a means of specifying the components of a system in a discrete event simulation. In DEVS formalism, one must specify Basic Models and how these models are connected together. These basic models are called Atomic models and larger models which are obtained by connecting these atomic blocks in meaningful fashion are called Coupled models. Each of these atomic models has inports (to receive external events), outports (to send events), set of state variables, internal transition,

external transition and time advance functions. Mathematically it is represented as 7-tuple system:

$$M = <X, S, Y, \delta int, \delta ext, \lambda, ta> \quad (1)$$

where X is an input set, S is set of states, Y is set of outputs, δint is internal transition function, δext is external transition function, λ is the output function, and ta is the time advance function.

The models description (implementation) uses (or discards) the message in the event to do the computation and delivers an output message on the outport and makes a state transition. DEVSJAVA [2], a Java-based implementation of DEVS formalism can be used to implement these atomic or coupled models.

1.3 Multi-Agent Simulation

In multi-agent systems, several agents work simultaneously to achieve a common goal. Agents can work autonomously or collaboratively. In multi-agent systems, there is a large amount of information passed among agents. This information also called a message contains knowledge which impacts the behavior of other agents. The message passing or communication protocol can be either direct or indirect. In direct message passing, the agents send information or messages directly to other agents. Here, the sending agent knows or should know the destination agent's address. Pure multi-agent system (MAS) architectures follow indirect message passing. A message router has the knowledge of other agents, which sends and receives the information. The message router understands the request and routes this message to appropriate agents, which can handle this request.

In the case of multi-agent systems with few agents, the message router may opt to broadcast the message to all the agents. In this case, the agents that handle that particular request, responds to the requestor. Other agents simply discard the message.

The multi-agent simulation architecture should possess properties such as,

- Modularity: Simulation architecture should allow the user to add or remove modules which affect the functionality of other modules of the simulation.
- Scalability: The impact of an increase in number of agents in the simulation architecture on the throughput of the simulation architecture should be minimal.
- Distributive problem solving capability: Distributing the problem or workload among agents which can reside on a single computer or on different computers, ensures a faster way to solve problems.

2 Virtual Laboratory (V-Lab®)

The modeling and simulation frameworks offer re-usability of models that help achieve better productivity, reduce error, decrease cost and development time, and simplify things to repeat past success. The models can be implemented as objects in an object-oriented framework, which provides several important features such as re-usability and data encapsulation (hidden from public view). The framework should act as a platform to organize the models in an accurate and efficient manner in order to realize any fully coupled system though simulation. Such a platform is what we call *Virtual Laboratory or V-Lab®* [3] for realizing fully coupled multi-agent system.

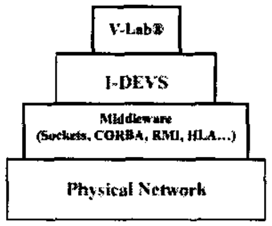

Figure 1. Layered Architecture of V-Lab®

Often the design of distributed simulations becomes large and complex. Applying a layered pattern (Refer Figure 1) to the design of a simulation breaks the simulation into several interconnected layers, each of which becomes more manageable than the simulation as a whole. Each layer has a distinct purpose and acts as a foundation for the layer above it. Furthermore, many different simulations require the same problems to be solved and the use of layers dramatically increases the amount of code that can be reused from simulation to simulation.

Intelligent DEVS (IDEVS) is the enhancement or enrichment to DEVS with different soft computing elements like fuzzy logic, neural networks, genetic algorithms, and stochastic learning automata.

2.2 Architecture of V-Lab®

The V-Lab® environment consists of four distinct software layers, as Figure 1 illustrates. Each of these layers fills a specific role in the simulation. The foundation of the simulation consists of the operating system and the network code needed to operate the networking hardware, which in turn allows machines to communicate over a network. Using this functionality, a middleware such as the Common Object Request Broker Architecture (CORBA) [4] acts to solve the problem of how to use the network to connect different portions of a simulation together. While middleware provides a useful tool for software interconnection, it does not provide the architecture needed to arrange components of a simulation into discrete structures. The Discrete Event System Specification (DEVS) provides this structure. Using the DEVS environment, V-Lab® defines an appropriate structure in which one can organize the elements of DEVS for a distributed agent based simulation. It separates the main components into different categories and defines the logical structure in which they communicate. It also provides the critical objects needed to control the flow of time, the flow of messages, and the base class objects designers will need to create their own V-Lab® modules.

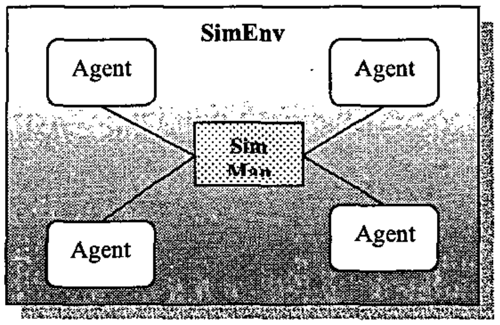

Figure 2. V-Lab® Architecture

3 I-DEVS

Soft computing, also called *computational intelligence*, is a consortium of tools for natural intelligence stemming from approximate reasoning (fuzzy logic), learning (neural networks, stochastic learning automata), optimization techniques (genetic algorithms, genetic programming), etc. Soft computing has been and is being extensively used in many applications such as robotics, manufacturing processes, control engineering, economics, software engineering, etc. We take advantage of simulation tools available and fuse it with these soft computing paradigms in a meaningful method to give us an *intelligent simulation tool*

Intelligent DEVS or IDEVS is fusion of DEVS and soft computing paradigms. Enhancement or enrichment to DEVS with different soft computing elements like fuzzy logic, neural networks, genetic algorithms, and stochastic learning automata gives different components of IDEVS like fuzzy-DEVS [5], NN-DEVS, GA-DEVS [5], SLA-DEVS [6], respectively. IDEVS provides users *set of libraries* for performing soft computing based simulation.

3.1 Neural Network DEVS

As an example of I-DEVS we consider here, the Neural Network-DEVS (NN-DEVS) implemented using Back Propagation Algorithm. Back Propagation Neural Network (BPNN) is a general Neural Network and it has been widely used in an abroad area. Back-propagation, which is also known as the generalized delta rule, is one of the most popular and widely investigated methods for training neural networks. It can be implemented in DEVS. There are two advantages of using Neural Network in Devs: handling partial lack of system understanding and creating adaptive models (models that can learn). It is mainly applied in three-areas [7]:

1. Concurrent simulation, where results of a Neural Network (NN) model are compared with results of a less realistic but validated common model to avoid a non expected behavior of the Neural-Net.
2. NN as sub-components of a global model, to model subsystems that would be hard to model commonly because of a lack of understanding.
3. Adaptive models, "models that can learn", according to an error feedback such model would be able to adapt runtime to situations that hasn't been taken into account.

Structure of BPNN

The most common network topology is multiple layers with connections only between nodes in neighboring layers. There are no connections between nodes located in a common layer. Its structure is presented in Fig. 3, where the number of hidden layers can be one or more than one.

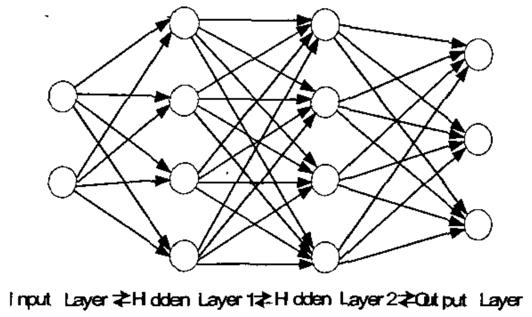

Figure 3. A Typical Neural Network with 2 Hidden Layers

There are two passes in BPNN:

Pass 1: Forward Pass - Present inputs and let the activations flow until they reach the output layer.

Pass 2: Backward Pass - Error estimates are computed for each output unit by comparing the actual output (Pass 1) with the target output. Then, these errors flow from the output layer to the hidden layers. Error estimates are used to adjust the weights in the hidden layer and the input layer.

The foundation of the back-propagation learning algorithm is the nonlinear optimization technique of gradient descent on the sum of squared differences between the actual output in the output nodes and the desired output. The detail about it can be found in several neural network books.

Implementation of BPNN in DEVS

A 4 layer BPNN can be implemented in DEVS, corresponding to the topology of Fig. 3. It is composed of input, hidden and output models. Each atomic model includes forward and backward computation (see Figure 4). It has training and testing phases. You can provide training data to inputs and set a stopping criterion to get a desired performance. After the training phase the trained weights can be used to test data. It can be extended to include more hidden layers by adding more hidden models. It can also be decreased to 3 layer BPNN if it is just composed of input and output models.

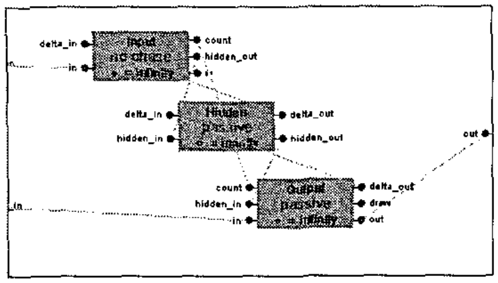

Figure 4. DEVS Atomic model implementing the 4-Layer NN

4 Experimental Results

4.1 Multi-agent Robotic Application

The simulation involves robots reaching a goal position from random initial point avoiding the obstacles. The obstacle information is defined in terrain model. The agents (robots here) navigate with the help of sensors, infra-red, GPS and compass, each of which are implemented as DEVS atomic model. The robots do not have global map information of the terrain, but local sensory information helps the robot to detect obstacles, accelerate, or decelerate, based on control algorithm in the controller module. NN-DEVS can be implemented as a controller module.

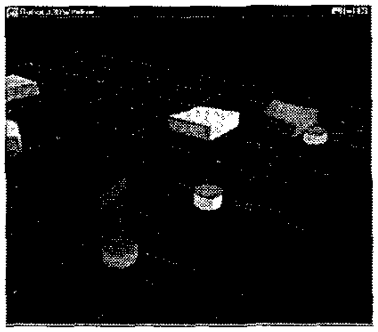

Figure 5. 3-D Robotic Simulation

4.2 Wireless Sensor network Application

Wireless sensor networks have been identified as one of the most important technologies for the 21st century [8]. With advancements in MEMS and wireless technology, small-sized sensors are ready for rapid deployment in military and commercial applications. However, development of such small sized, low energy consuming and information processing entities is still a technical challenge. Discrete event simulation of a small wireless sensor network is demonstrated in this paper using DEVS formalism.

Simulating sensor networks can be done at two levels: Node module level and network module level. At the node level, individual sensor nodes are modeled and simulated. At the network module level, entire sensor networks can be modeled, with detailed simulation of node placement, self organization, routing algorithm, and so on. Since each of the sensor nodes sense the environment using sensors and perform limited computation, the sensor network can be simulated as a multi-agent system, where, each individual nodes of the network are treated as agents performing autonomous tasks. The DEVS formalism helps us to model these sensor nodes as agents.

Implementation details
In this research paper, we would like to simulate the node module level of the sensor networks. Each of the sensor nodes are implemented as DEVS coupled model which consists of *DEVS atomic* models coupled together in meaningful fashion. The atomic models perform the finest possible functionality of a sensor node. The architectural view of a sensor node is given in figure 6.

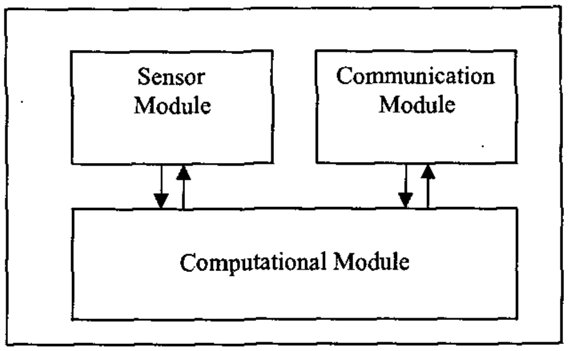

Figure 6. Sensor node

Each of the modules within the sensor node (or agent) is simulated as DEVS atomic model. The sensor module is implemented as DEVS atomic model which accept ID, X_s, Y_s, and distance (d) as inputs. The output of the sensor block is *voltage* which is calculated from the distance input in the look-up table. The computational module computes the appropriate task the sensor node has been deployed for. In our simulation, computational node, hold a buffer to send the sensed information to the communication module. Any input received from the communication module is also buffered for routing to other sensor nodes or agents. The detailed functionality of all the modules is explained in the next section describing a sample scenario.

Sample scenario

The scenario is to simulate the communication of a signal sensed at one sensor node to the processing node involving multiple hops. In wireless sensor networks, each of the sensor nodes does not know the existence of other nodes. In order to simulate a network, we make an assumption that each node is couple to few other nodes which it can broadcast the signal to. This assumption can be easily eliminated by using V-lab® architecture of having a central manager (SimMan), which would hold the communication information. Specifically, SimMan holds a table with the list of all the nodes and all nodes which each node can transmit or broadcast the signal to. However, to simulate a simple scenario, we will not use SimMan and assume that each node *x* is coupled to *m* other nodes for communication.

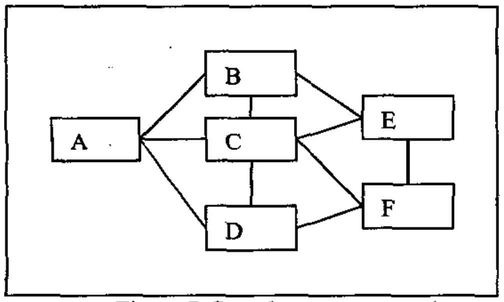

Figure 7. Sample sensor network

When a signal is injected at node A, the transmission is carried to all the nodes B, C and D. Each of these nodes transmit the signal to node E and F and back to node A (Refer Figure 7).
Since there are multiple copies of message or signal at E and F, we use message IDs to identify the redundant message. Any copy of same message appearing at node is dropped and is not further broadcasted.
When the broadcast from B, C and D are received, a signal is injected at the sensor module of E indicating that a message has been sensed by the sensor module at E. This would generate a new message ID. Each of the sensed messages is put to a sensor buffer in the computational module and is flagged with a new ID. Any message from the communication module is put in a communication buffer in the computation module and it retains the same ID as is broadcasted. Figure 8 gives the second level architectural view of the sensor node.

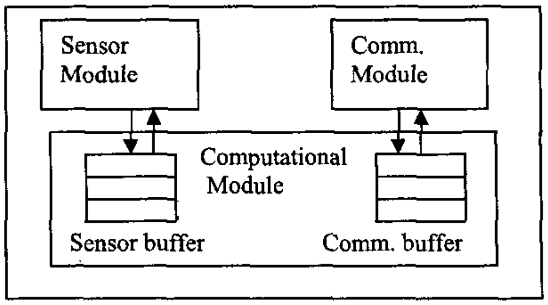

Figure 8. Simulating the node modules

5 Conclusions

In this paper we discussed about modular and distributable framework for multi-agent simulation called V-Lab®, built on top of DEVS formalism. V-lab® provides a layered and modular approach when the simulation becomes large and complex. As a future work, wireless sensor networks can be simulated within the V-Lab® infrastructure.

References

[1] Zeiglar, B. P., Praehofer, H., Kim, T.G., "Theory of Modeling and Simulation", Second edition, Academic Press, Boston, 2000.

[2] Zeiglar, B.P. and Sarjoughian, H., "Introduction to DEVS Modeling and Simulation with JAVA: A Simplified Approach to HLA-Compliant Distributed Simulations", ACIM, http://www.acims.arizona.edu

[3] El-Osery, A., J. Burge, M. Jamshidi, et al. "V-Lab® – A Distributed Simulation and Modeling Environment for Robotic Agents – SLA-Based Learning Controllers," *IEEE* Transactions on Systems, Man and Cybernetics, Vol. 32, No. 6, pp. 791-803, 2002

[4] The Object Management Group, "CORBA BASICS," electronic doc, www.omg.org/gettingstarted/corbafaq.htm

[5] Sridhar, P, Sheikh-Bahaei, S., Xia, S., Jamshidi, M., "Multi-agent simulation using Discrete event and soft computing methodologies", Intl. conference on Systems, Man and Cybernetics, IEEE, Washington D.C., 2003.

[6] Jamshidi, M., Sheikh-Bahaei, S., Kitzinger, J., Sridhar, P., et al, "A Distributed Intelligent Discrete-Event Environment for Autonomous Agents Simulation",

Chapter 11,"Applied Simulation" , Kluwer Publications 2003.

[7] Filippi, J-B., Bisgambiglia, P., Delhom, M., "Neuro-Devs, an Hybrid Environment to Describe Complex Systems", ESS '2001 13th European Simulation Symposium and Exhibition.

[8] Chong, C-Y., Kumar, S., "Sensor Networks: Evolution, Opportunities, and Challenges", Invited paper, Proceedings of the IEEE, Vol. 91, No. 8, Aug 2003.