

V-LAB® – A VIRTUAL LABORATORY FOR INTELLIGENT AGENTS

M. Jamshidi, S. Sheikh-Bahaei, J. Kitzinger, P. Sridhar, S. Xia, Y. Wang, J. Liu, T. Song, Beatty, U. Dole, E. Dong, P. Swiderski, B. Masefield, E. Tunstel, Jr¹, M. Akbarzadeh², A. Osery³ and M. Fathi⁴

Department of Electrical and Computer Engineering and Autonomous Control Engineering (ACE) Center, University of New Mexico, Albuquerque, New Mexico 87131
moj@cybermesa.unm.edu & jamshidi@ece.unm.edu

A virtual laboratory is being created at University of New Mexico's ACE Center (in collaboration with Arizona Center for Integrated Modeling and Simulation) to simulate and emulate such problems like robotic colonies, recognisance, monitoring and homeland security with both software and hardware implementation. The project deals represents a fusion between discrete-event systems specification (DEVS) and intelligent tools from soft computing. DEVS provides a robust and generic environment for modeling and simulation applications employing single workstation, distributed, and real-time platforms. Soft computing is a consortium of tools for natural intelligence stemming from approximate reasoning (fuzzy logic), learning (neural network or stochastic learning automaton), optimization (genetic algorithms and genetic programming), etc. The outcome of this fusion is what is called "Intelligent DEVS", called IDEVS here. IDEVS is an element of a virtual laboratory, called V-Lab®, which is based on distributed multi-physics, multi-dynamic modeling techniques for multiple platforms. The chapter will introduce IDEVS and V-Lab® and a theme example for a multi-agent simulation of a number of robotic agents with a slew of dynamic models and multiple computer work stations.

1. V-LAB® OVERVIEW AND ARCHITECTURE

The V-Lab® environment consists of 4 distinct software layers, as Figure 1 illustrates, and 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) [1] acts to solve the problem of how to use the network to connect different portions of a simulation together. While CORBA 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) [2-4] provides this structure. Using the DEVS environment, V-Lab® defines an appropriate structure in which to organize the elements of DEVS for a distributed agent based simulation. It separates the main components into different categories and defines the logical structure

¹-Author is with the Jet Propulsion Laboratory, Cal Tech, Pasadena, CA

²-Author is with the Department of Electrical and Computer Engineering at Ferdowsi University at Mashad, IR.

³-Author is with the Department of Electrical Engineering at New Mexico Institute of Technology, Socorro, NM.

⁴-Author is with the Department of Mechanical Engineering at Florida State University, Tallahassee, FL .

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.

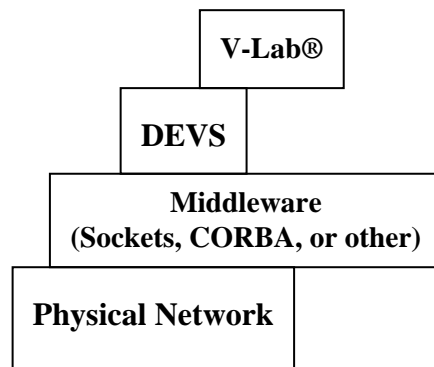


Figure 1- Distributing simulation layers using DEVS.

Just as the middleware (CORBA or Sockets) defines the core functionality of module intercommunication, and DEVS defines the hierarchical and compositional organization of the modules, V-Lab® defines the logical structure in which to implement these modules. Each successive layer defines a more specific organizational structure for the simulation than the last. However, each layer also restricts the domain of problems that can be addressed by the architecture. CORBA may be a valid option for an application that has a user interface communicating with a spreadsheet, but DEVS is not. DEVS may be a valid option for constructing a simulation with millions of cells, but V-Lab® is not. Specifically, V-Lab® is an architecture that defines a logical structure for simulations with a relatively small number of agents interacting in complex ways. Likewise, in DEVSJAVA 2.7 the user has few restrictions when specifying inter-module communication whereas with V-Lab®, multi-agent simulations require conformance to V-Lab®'s structured communication protocol. Following the rules arising from the architecture defined by V-Lab® allows simulation designers to create a simulation that is modular, extendable and allows for the re-use of pieces of a simulation in future simulations by providing a level of indirection between components of the simulation. Critical backbones of V-Lab® will be tools from soft computing (SC) paradigms like fuzzy logic (FL), neural networks (NN), genetic algorithms (GA) and stochastic learning automaton (SLA). DEVS and SC together constitute what we call IDEVS, intelligent discrete-event systems specification to be detailed in [5]. The design of V-Lab® was introduced by El-Osery, et al. [6].

The Discrete Event System Specification (DEVS) modeling and simulation environment, DEVSJAVA®, was developed by the Arizona Center for Integrative Modeling and Simulation, headed by Zeigler [2-4]. It was created to provide a robust and generic environment for modeling and simulation applications employing single workstation, distributed, and real-time platforms. The DEVS environment expands the capabilities of more generic distributed architectures such as CORBA. Whereas CORBA acts as the core functionality that allows different modules on different machines to communicate, DEVS defines the structure in which these modules are created and communicate with each other. The DEVSJAVA 2.7 environment provides Java classes that encapsulate all the functionality that is needed to create a module which is fully capable of being connected to other modules in a meaningful relationship, regardless of which machines these modules are located on. Details of DEVS will be referred to literature on the subject [5,7 due to shortage of space.

1. INTELLIGENT DEVS

One of the main objectives of V-Lab® is to enhance DEVS with the tools available by soft computing, e.g. fuzzy logic, genetic algorithms, neural networks and stochastic learning automaton by introducing them in discrete-event simulation environment. In this section four paradigms are introduced within DEVS. We denote this intelligent DEVS or IDEVS. Here we present a DEVS implementation of fuzzy logic, genetic algorithms, stochastic learning automaton and neural networks.

Fuzzy-DEVS. One of the more popular members of the soft computing consortium is fuzzy logic, which has been implemented in DEVSJAVA 2.7¹. One of the more important applications of fuzzy logic is fuzzy logic control. A fuzzy logic controller consists of three operations: (1) fuzzification, (2) inference engine, and (3) defuzzification. The input sensory (crisp or numerical) data are fed into fuzzy logic rule based system where physical quantities are represented into linguistic variables with appropriate membership functions. These linguistic variables are then used in the antecedents (IF-Part) of a set of fuzzy “IF-THEN” rules within an inference engine to result in a new set of fuzzy linguistic variables or consequent (THEN-Part) [9].

A typical Mamdani rule can be composed as follows

$$\text{IF } x_1 \text{ is } A_1^i \text{ AND } x_2 \text{ is } A_2^i \text{ THEN } y^i \text{ is } B^i, \text{ for } i = 1, 2, \dots, l$$

where A_1^i and A_2^i are the fuzzy sets representing the i th-antecedent pairs, and B^i are the fuzzy sets representing the i th-consequent, and l is the number of rules.

In order to simulate a fuzzy-logic controller using DEVS, the following atomic and coupled models are designed: (Inputs x_1 and x_2 are crisp values, and max-min inference method is used. Also rules are assumed to be disjunctive.)

Antecedent Membership Function. This atomic model receives a crisp value as input and generates its fuzzified value as output. This model can accept various membership functions like triangular, singleton, left-shoulder, right-shoulder, trapezoidal, etc. as a parameter.

Connectives. There are two connectives : AND , OR. They simply take the minimum, maximum of two fuzzy inputs A and B, respectively.

Consequent Membership Function. This model receives the fuzzy value either from an Antecedent Membership Function or from a Connective and generates a truncated membership function (fuzzy set). This model also can accept various different membership functions as a parameter.

Rule. This coupled model consists of the above atomic models. It takes the fuzzy rule as a parameter and automatically makes the proper coupling among those atomic models to simulate the fuzzy rule. For example the rule “IF x is A OR y is B THEN z is C” will have the form in Figure 2.

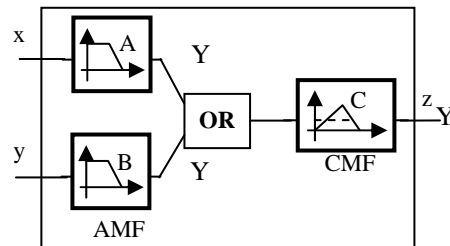


Figure 2. A DEVS model for a typical fuzzy rule” IF x is A OR y is B THEN z is C”

A similar atomic model can be created for logical operations like union, intersection, compliment, etc. In references [5] other elements of soft computing like Genetic Algorithms, Neuro-computing, Stochastic learning automaton, etc. are already fused into DEVS environment.

A THEME EXAMPLE SIMULATION

The objective of this example is to demonstrate and test the various modules of the IDEVS within the proposed V-Lab®. architecture in a multi-physics multi-agent distributed simulation. This example allowed one to test some soft computing methods for autonomous agents in DEVSJAVA® 2.7 environment.

¹ Other extensions of DEVS including Fuzzy versions can be found in reference [8].

Autonomous control algorithms are used to control the maneuvering of the rovers and avoid the obstacles to reach the goal position.

These modules are implemented as atomic or coupled modules in DEVS. Figure 3 presents the simulation block diagram of the system.

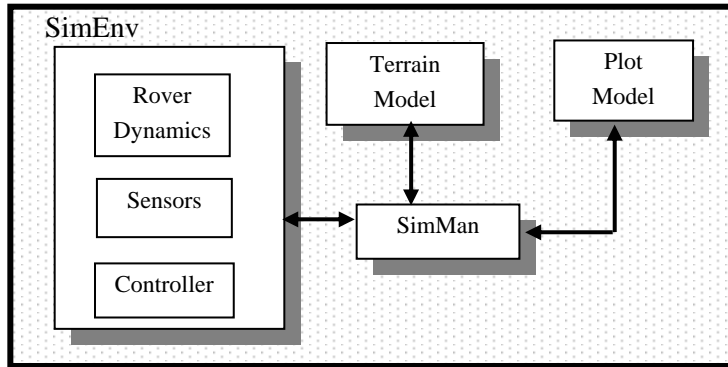


Figure 3. Schematic of an Example Simulation

SimMan/SimEnv represent the basic components of the V-Lab® multi-agent simulation system [6]. SimEnv is actually the top-level DEVS coupled model of a V-Lab® simulation. The user instantiates a SimEnv with the name of a configuration file. The configuration file contains the agents that start out in the simulation. SimEnv instantiates one model (atomic or coupled) for each agent specified in the configuration file. It also instantiates SimMan – another component of SimEnv. The connections between these models reflect the *decoupled nature of a V-Lab® simulation*, i.e. all agents are only connected to SimMan.

SimMan acts as a relay mechanism – it relays event messages to and from agents as they make requests of each other. The agents do not have to know the identity of the other agents. The way that these agents communicate is via the type of the request message. Specified in each agent model is a registration-list of request-types that the model recognizes. When any other agent makes a request of this type, it is relayed by SimMan to this agent. Also specified in the registration-list is a timeout value for each request-type. If the number is positive, then the agent is expected to respond to the request in the amount of (simulation) time specified. Figure 4 shows the simulation result on Java Frame which shows the position of the obstacles and rover (with 3 sensors). The Java Frame and the motion plot frame are kicked-off by the *Plot Model*.

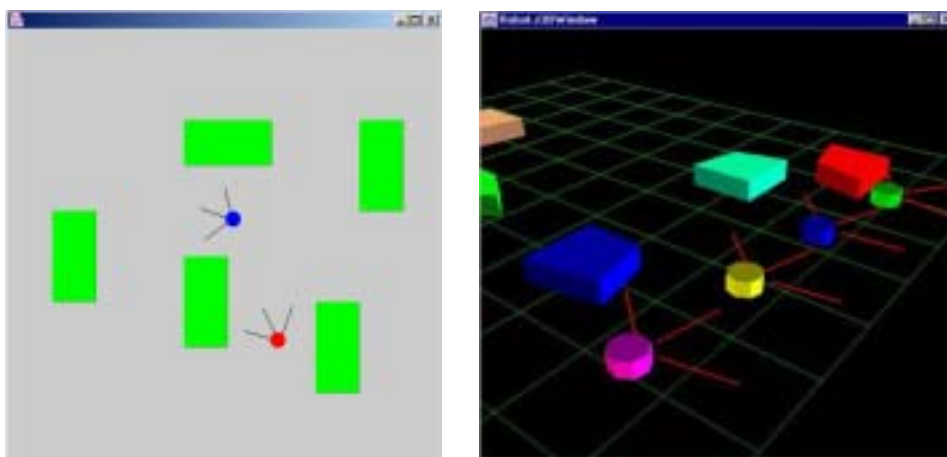


Figure 4. Simulation Result of Two-Rover 2-D Case and 4-rover 3D Case for the Theme Example

V-Lab architecture allows distributed simulation of multi-agent systems. The example given previously of one or more rovers navigating among a set of obstacles was tested on a single host. That is, all models and the classes representing them existed on one PC. The natural extension of this would be to allow distributed

simulation. This means putting different component models on different hosts and somehow coordinating the events between them. Some of the reasons for doing distributed simulation might be to increase performance or perhaps to facilitate group development of component models without giving up intellectual property.

GenDEVS and Sockets described elsewhere [6] is an overview of how distributed simulation with sockets works. The process is as follows: first, the user starts a coordinator on a server host. It must be given a coupled model that serves as the structural description of the simulation. This overall model simply contains the names of the components and their interconnections. (As a note, none of the components on the server need contain any functionality, i.e. the coordinator only needs a structural summary of the components. The model functionality is only executed on the clients as described below.)

Conclusions

V-Lab® architecture allows users to test different types of control algorithms for multi-agent simulation using DEVS. DEVS-Fuzzy, DEVS-GA, DEVS-NN are emerging ideas in the field of simulation which would make DEVS a standard for discrete event simulation and V-Lab® a standard for writing and testing multi-agent autonomous control algorithms.

Using the architecture of Figure 1 to build a simulation allows a developer to avoid much of the work involved with creating a distributed simulation. Process communication, handled by Sockets, supports DEVS, a well-established methodology for creating dynamic system objects and their communication. A structure specific for intelligent simulations is provided by V-Lab®. Furthermore, since V-Lab® uses a layered pattern in its design, if any of the layers becomes obsolete in the future it can be replaced without having to replace the entire architecture. With this design, V-Lab® offers a flexible and powerful approach to creating simulations that allows for the reuse and modularity of simulation components. Recent developments in DEVSJAVA® support model continuity – the ability to employ the same DEVS models that were developed in simulation form without change in actual real-time execution [10]. This is a capability that could significantly reduce the overall development time for intelligent systems.

Acknowledgements. The authors gratefully thank NASA Ames Research Center the support of this work. The cooperation of Arizona Center for Integrate Modeling and Simulation (ACIMS) is sincerely appreciated. IDEVS in V-Lab® is being developed under a cooperative software agreement between ACE and ACIMS.

References

1. The Object Management Group, "CORBA BASICS," electronic document, <http://www.omg.org/gettingstarted/corbafaq.htm>
2. Cho Y. K., *RTDEVS/CORBA: A Distributed Object Computing Environment For Simulation-Based Design Of Real-Time Discrete Event Systems*, Doctoral Dissertation ECE Dept., Univ. of Arizona, 2001.
3. Arizona Center for Integrative Modeling and Simulation, "DEVSJAVA 2.7software", <http://www.acims.arizona.edu/SOFTWARE/software.shtml>
4. Lee. J.S., B. P. Zeigler and S.M. Venkatesan, "Design and Development of a Data Distribution Management Environment", *Simulation*, 2001(77)1-2, July 39-52
5. Jamshidi, M., S. Sheikh-Bahaei, J. Kitzinger, P. Sridhar, S. Xia, Y. Wang, J. Liu, E. Tunstel, Jr , M. Akbarzadeh , A. El-Osery , M. Fathi, X. Hu , and B. P. Zeigler "A Distributed Intelligent Discrete-Event Environment for Autonomous Agents Simulation", Chapter 11, *"Applied Simulation"*, Kluwer Publications 2003.
6. El-Osery, A., J. Burge, M. Jamshidi, A. Saha, M. Fathi and M. Akbarzadeh-T. "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.
7. M. Jamshidi, S. Sheikh-Bahaei, J. Kitzinger, P. Sridhar, S. Beatty, S. Xia, Y. Wang, T. Song, J. liu, E. Tunstel, Jr., M. Akbarzadeh, P. Lino, U. Dole, A. El-Osery, M. Fathi, X. Hu and B. P. Zeigler, "V-Lab – A Distributed Intelligent Discret-Event Environment for Autonomous Agents Simulation," *Intelligent Automation and Softy Computing*, Volume 9, No. 3, 2003.

8. Zeigler B. P., Kim T.G. and Praehofer H., *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2nd Edition, Academic Press, Boston, 2000.
9. Jamshidi M., A. Zilouchian, *Intelligent Control Systems using Soft Computing Methodologies*, CRC Press, Boca Raton, FL, 2001.
10. Hu, X. and Zeigler B.P., An Integrated Modeling and Simulation Methodology for Intelligent Systems Design and Testing, Proc. of PERMIS'02, Gaithersburg, Aug, 2002