Multi agent Simulation using Discrete Event and Soft-computing Methodologies*

Prasanna Sridhar
Center for Autonomous Control Engineering (ACE) and Department of Computer Science, University of New Mexico, Albuquerque, NM, USA
prasanna@cs.unm.edu

Shahab Sheikh-Bahaei
Center for Autonomous Control Engineering (ACE) and Department of Electrical & Computer Engineering, University of New Mexico, Albuquerque, NM, USA
shahab@unm.edu

Shan Xia
Center for Autonomous Control Engineering (ACE) and Department of Electrical & Computer Engineering, University of New Mexico, Albuquerque, NM, USA
xiashan@unm.edu

Mo Jamshidi
Center for Autonomous Control Engineering (ACE) and Department of Electrical & Computer Engineering, University of New Mexico, Albuquerque, NM, USA
jamshidi@eece.unm.edu

Abstract - *With the emerging applications of multi-agent systems, there is always a need for simulation to verify the results before actual implementation. Multi-agent simulation provides a test bed for several soft-computing algorithms like fuzzy logic, learning automata, evolutionary algorithms, etc. In this paper we discuss the fusion of these soft-computing methodologies and existing tools for discrete event simulation (DEVS) for multi-agent simulation. We propose a methodology for combining the agent-based architecture, discrete event system and soft-computing methods in the simulation of multi-agent robotics and network security system. We also define a framework called Virtual Laboratory (V-Lab®) for multi-agent simulation using intelligent tools.*

Keywords: Soft-computing, DEVS, IDEVS, V-lab®, Fuzzy-DEVS, GA-DEVS.

1 Introduction

Most of the AI-based simulation is based on the concepts of agents. Agents are entities (robots, software entities, animals, etc) which can sense or perceive the environment and act upon the environment based on some control paradigms. Multi-agents are agents that can work together (collaborative) or act *autonomously* in their environment to realize a set of goals. Applications of multi-agents are many. Such as the exploration of the Martian landscape, rovers can be used to quickly map an unstructured environment, to move an object larger than any single rover could move or to cooperatively navigate in rough terrain. Using multiple agents also allows for the failure of a single agent without the failure of the entire mission, ensuring *reliability*. *Modularity* is also a key feature of agent based simulation. We use Discrete Event System Specification (DEVS) [10][11] modeling for multi-agent simulation using some of the soft computing methods.

1.1 DEVS

Discrete Event System Specification (DEVS) 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 as shown in Figure 1. 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:

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle \quad (1)$$

where X is an input set, S is set of states, Y is set of outputs, δ_{int} is internal transition function, δ_{ext} is external transition function, λ is the output function, and ta is the time advance function.

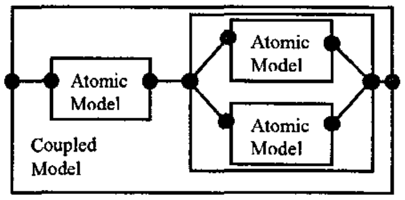

Figure 1. Sample DEVS Atomic and Coupled Models

* U.S. Government work not protected by U.S. copyright

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 [1], a Java-based implementation of DEVS formalism can be used to implement these atomic or coupled models.

1.2 Soft Computing

The word soft computing was coined by L. Zadeh, which is based on concepts such as human-like reasoning, learning in uncertain environment, probabilistic reasoning, to give robust solution to problems, in low cost. 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. Simulation is needed in advance, before the implementation and testing of these soft computing paradigms in real systems. An example of this is the automation and control of robotic agents. 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*.

2 Intelligent DEVS

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, NN-DEVS, GA-DEVS, SLA-DEVS, respectively. Fusion of soft computing methodologies with simulation tools such as DEVS provides a novel and systematic way of handling time-dependent parameters without altering the essential functionality and problem-solving capabilities of soft computing elements. In this paper we will see two such elements of IDEVS namely GA-DEVS and Fuzzy-DEVS and application of Fuzzy-DEVS in simulation of computer network security system.

2.1 GA-DEVS

Genetic Algorithm (GA) is an element of evolutionary computation, which is a rapidly growing area of soft computing. The continuing price/performance improvements of computational systems have made GAs attractive for some types of optimization. In particular, genetic algorithms work very well on mixed (continuous *and* discrete), combinatorial problems. They are less susceptible to getting 'stuck' at local optima than gradient search methods [4]. Although GA tends to be computationally expensive, one can use some methods to accelerate its process.

There are four main parts in the GA process, namely, the problem representation or encoding, fitness or objective function definition, fitness-based selection, and evolutionary reproduction of candidate solutions (individuals or chromosomes). So we need to define an encoding method, fitness function, select method, and reproduction method as well as criteria rules for the GA formulation. Figure 2 shows the basic cycle of a GA.

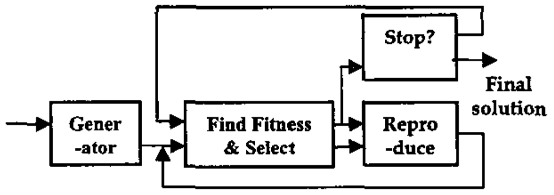

Figure 2. Basic cycle of a genetic algorithm

The basic GA cycle is implemented via atomic and coupled models of DEVS. Figure 3 shows a GA-DEVS implementation. An *initial* model was added to collect required initial conditions. In every generation, we can get not only the current best fitness value (*fm*) but also the average and deviation of fitness value (*fa, fd*) of the population. So we can find the tendency of the process by observing the history of fitness during evolution.

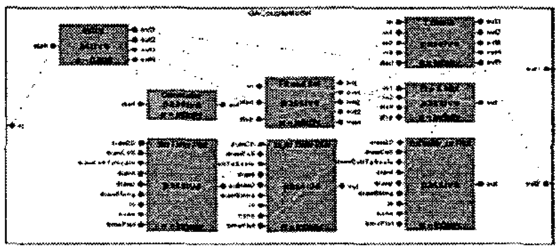

Figure 3. GA-DEVS implementation

The functionality of the framework proposed is tested with several examples. In the following example we use flexible reproducing operators to accelerate the GA process, i.e., modifying the probability of crossover (*Pc*) and probability of mutation (*Pm*) according to the deviation fitness of the population. Based on the resulting performance analysis of GA-DEVS example-simulations, we found that properly adjusting *Pc* and *Pm* during the progress will improve the algorithm. However, GA is time consuming, so sometime they didn't reach the global optimum during desired simulation time.

A simulation result of *fm* and *fa* with generations is sketched in figure 4. It can be seen that the maximum fitness value never goes down because of using the elitism strategy and the average fitness curve also keeps an upward tendency. When the average fitness goes very close to max fitness value, which means, the diversity is almost lost, it will be good to have more mutation. This also verifies that flexible *Pc* and *Pm* will benefit the progress and we still need to avoid the scaling problem.

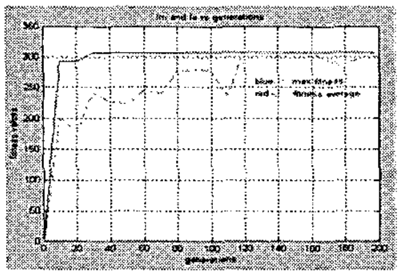

Figure 4. Average & max fitness values versus generations

The problem depicted above is to find the real value of *x* which maximizes the function

$$f(x) = (-x^2+255x)/42.5 \quad (2)$$

for values of x in the range [0.00...255.00]. The best individual is x=127.5. The tolerant error amount of x is 1.5 and for y is *|382.4471-382.50|=0.0529*. Linear transformation $f'=af+b$ with a=0.8, b=4 is used, so $f'm = 0.8*382.5+4 = 310$.

This GA-DEVS simulation example has an advantage to perform reasonably well within the limit of total generation of GA. GA-DEVS will be a tool to optimize a fuzzy or a classical controller for multi-agent simulations.

2.2 Fuzzy-DEVS

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) [5]

A typical Mamdani rule can be composed as follows

IF x_1 is A_1^i AND x_2 is A_2^i THEN y^i is B^i, for $i = 1, 2, ..., l$ where A_1^i and A_2^i are the fuzzy sets representing the ith-antecedent pairs, and B^i are the fuzzy sets representing the i^th^-consequent, and l is the number of rules.

We define *DEVS-Fuzzifier* as follows:
A *DEVS-Fuzzifier* is an atomic DEVS model:

$$FZ = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta, \mu) \quad (3)$$

where, the input set, X, is usually a set of real numbers, to be fuzzified. The output set Y = [0,1], since the output of this atomic model is a fuzzy value. S is a sequence of fuzzified input values: $S=\{s_i \mid s_i = \delta_{ext}(q,s_{i-1},x)\}$. $\delta_{ext}(q,s,x)=\mu(x)$, where μ is the membership function associated with this fuzzifier. $\delta_{int}(s)$ and $\lambda(s)$ are the identity functions. ta (s) =0 , since there is no time advance in this model. $\mu(x)$ is the membership function associated with this fuzzifier.

This atomic model represents an antecedent membership function.

A *consequent membership* function can be defined as:

$$CMF=(X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta, \mu) \quad (4)$$

where, X is a set of fuzzy values: X=[0,1]. The output of this model is a fuzzy set, so Y is a set of fuzzy sets. Y = $\{\Sigma \mid \Sigma$ is a fuzzy set$\}$. S is a sequence of discrete fuzzy sets (membership functions). $S=\{\ldots, \Sigma_{i-1}, \Sigma_i, \Sigma_{i+1}, \ldots\}$

$$\delta_{ext}(q,s,x)= \sum \frac{\min(\mu(a_i),x)}{a_i} \quad (5)$$

where, μ is the membership function associated with this model. This function in fact cuts those values of the fuzzy set $\Sigma\mu(a_i)/a_i$ which their membership values are greater than x (see Figure 5). $\delta_{int}(s)=s$, $\lambda(s)=s$ and ta=0.

In a typical fuzzy rule connectives are used to make connection between antecedent pairs. An "AND" connective takes the minimum, while an "OR" connective returns the maximum value of two pairs [5].

Connectives are defined in DEVS as atomic models:

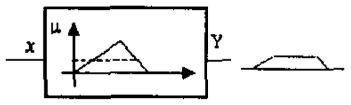

Figure 5. A typical consequent membership function (CMF) model

$$CNTV=(X_1, X_2, \ldots, X_n, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta, \mu) \quad (6)$$

where, $X_1=\ldots= X_n=[0,1]$ are input sets. (A connective can have n inputs.), Y = S = [0,1], $\delta_{int}(s) = s$, $\lambda(s) = s$, ta = 0, $\delta_{ext}(q,s,x_1,\ldots x_n) = \min(x_1,\ldots,x_n)$, in case of "AND" and, $\delta_{ext}(q,s,x_1,\ldots x_n) = \max(x_1,\ldots,x_n)$, in case of "OR".

A fuzzy *rule* can be composed as a *coupled* model using the above three *atomic* models. For instance, consider the following fuzzy rule: "IF x is A OR x is B THEN z is C".

Figure 6 shows how this fuzzy *rule* can be composed using two *Fuzzifiers*, one *Connective* and one *CMF* (consequent membership function).

In order to fully implement a Fuzzy Logic Controller, a *Defuzzifier* is needed, which can be defined in the same way. i.e. takes fuzzy sets as input and calculates the defuzzified value [5].

$$DeFZ=(X_1, X_2, \ldots, X_n, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta) \quad (7)$$

where, Inputs of this atomic model are fuzzy sets so
$X1=X2=\ldots=X_n=\{x \mid x \text{ is a fuzzy set}\}$
Output is a real number, $Y = \Re$,
$S = \Re$, $\delta_{int}(s) = s$, $\lambda(s) = s$, $ta = 0$,
$\delta_{ext}(q,s,x_1,\ldots x_n) = \text{Center\_of\_Gravity}\,(x_1 \cup x_2 \cup \ldots x_n)$

Center_of_Gravity(x) is a function which calculates the defuzzified value of fuzzy set *x* using *center of gravity* method [5, page 193] :

$$Center\_of\_Gravity\,(x) = \frac{\int x\mu(x)dx}{\int \mu(x)dx} \quad (8)$$

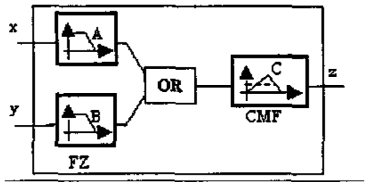

Figure 6. A DEVS model for a typical fuzzy rule "IF x is A OR y is B THEN z is C"

3 Virtual Laboratory

V-Lab® [2] stands for Virtual Laboratory for autonomous agents. The V-Lab® environment consists of 4 distinct software layers, as Figure 7 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 Remote Method Invocation (RMI) [9], Common Object Request Broker Architecture (CORBA), High Level Architecture (HLA) [8] or even sockets acts to solve the problem of how to use the network to connect different portions of a simulation together. Using the IDEVS environment, V-Lab® defines an appropriate structure in which one can organize the elements of IDEVS for a distributed agent based simulation. 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.

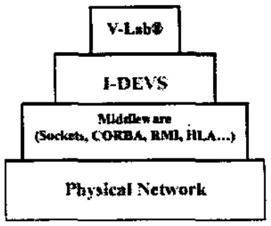

Figure 7. Layered Architecture of V-Lab®

3.1 Architecture of V-Lab®

Modular architecture of V-Lab® is as presented in Figure 8. The different modules of V-Lab® are the agents which are implemented as DEVS or IDEVS atomic or coupled models. The environment provides plug-and-play for agents to be added and removed without distracting the functionality of other agents. This modular architecture is similar to the *mediator* design pattern [4].

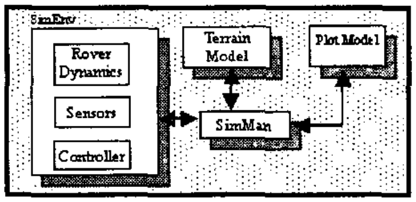

Figure 8. Modular Architecture of V-Lab®

Essential parts of V-Lab® are the SimMan (Simulation Manager) and SimEnv (Simulation Environment). SimEnv is the high level coupled model. V-Lab Simulation kicks off by starting SimEnv which instantiates all other modules. SimMan acts as message relay for all other modules. With SimMan, V-lab has the property of separability, i.e., the modules do not know the existence of other modules. The modules simply *publish* the messages that they can handle. All other modules *request* or *subscribe* by sending appropriate messages to the SimMan.

4 Applications

4.1 Computer network security simulation

As an application of IDEVS components, we demonstrate discrete event simulation of computer network security system using Fuzzy-DEVS. We use combination of *crisp* and *fuzzy* rules to simulate a network security system which encompasses both *prevention* and *detection* features.

Prevention systems are network security systems which block unauthorized access to local network, such as firewalls, packet filters, etc. *Detection systems* are systems which detect misuse (or intrusion) but do not necessarily handle the prevention of such a misuse. Examples are host based and network based intrusion detection (NIDS), etc. The application discussed here focuses on the simulation of combination of prevention and detection systems using Fuzzy-DEVS, which employs similar modular approach as in V-Lab®.

4.1.1 Implementation Details

In this paper, for prevention system functionality we simulate packet filtering firewall such as *iptables* [7] found in Linux. The firewall rules defined for filtering is crisp. The rules are written in a configuration (ConfigRules) file which is read each time when the packet arrives. Typical policy rules are shown in figure 9.

```
//Src_ip Dest_ip  Src_port  Dest_port  Action
ANY  127.0.0.1    6000    23    DENY
23.34.45.56    64.1.2.3   1024    21    ACCEPT
```

Figure 9. Filtering Rules

The entire simulation is based on multi-agent architecture; in which each agent perform specific filtering functionality. We have three agents TCP agent, UDP agent and ICMP agent for handling TCP, UDP, and ICMP protocol packets. A packet sensor looks into the packet header to determine the protocol and passes the packet to specific agent which can handle those packets. The execution of the policy rules for each packet's acceptance or rejection is done by the TCP and UDP agents. For the purpose of simulation, we implement several DEVS atomic model, each of which can generate packets at particular time period with packet generation rate as input to these models (Refer figure 10).

Several intrusion detection classifications can be done such as: Invalid logins, abnormal connections, anomalies, suspicious modification of files, denial of service, protocol violation etc. We demonstrate here a particular case of intrusion which is denial of service attack. Denial of Service (DoS) is an attack wherein the intruder or hacker sends large number of packets (ICMP echo packets or SYN packets) in small amount of time. We employ *temporal fuzzy rules* in ICMP agents to detect such large number of packets within short interval of time. While the TCP and UDP agents perform the filtering (prevention system) of network packets using crisp rules, ICMP agent functions as detection system employing temporal fuzzy rules to detect ICMP ECHO denial of service attacks. The fuzzy rules can also be extended to TCP SYN flooding by introducing the rules in TCP agent. The ICMP agent is coupled model, which helps to detect DoS attacks. The IP address of the intruder is assumed to be known here. So there is one intruder attacking our network with several ICMP ECHO packets.

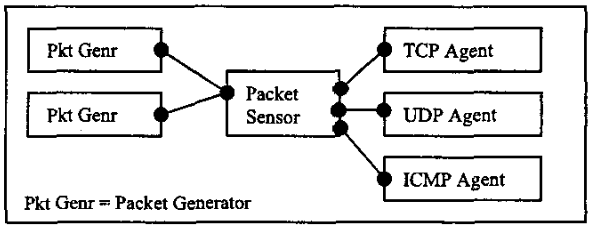

Figure 10. Agent based Intrusion detection system

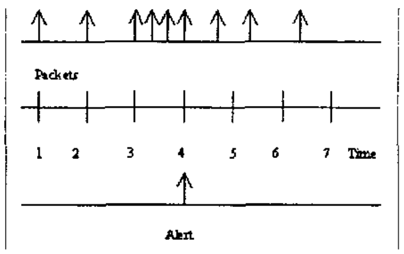

Figure 11. Generation of Alert due to DoS attack

From figure 11, we can infer that when packet arrival rate increases, there is an intrusion. Since we assumed only one intruder sending lot of ping (ICMP request) packets within short interval of time, there is alert for intrusion from the system. Simple temporal fuzzy rules can be written, such as,

If packet_arrival_rate is HIGH in *Last_2_seconds*, then ALERT is HIGH

If packet_arrival_rate is LOW in *Last_2_seconds*, then ALERT is ZERO.

The membership function for packet arrival rate is as shown in figure 12. Note the packet rate numbers are experimental.

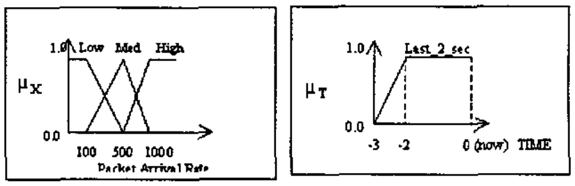

Figure 12. Membership functions for packet arrival rate (μ_x) and Last_2_seconds (μ_T)

Membership functions for output 'Alert' are as shown in figure 13.

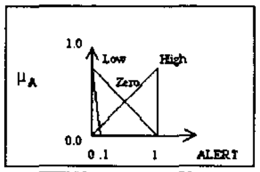

Figure 13. Membership function for output ALERT

The degree of fuzziness in temporal fuzzy logic obtained by following formula [6]:

$$\mathrm{DOF} = \max_{t_k \in \mathrm{supp}_T} (\min(\mu_x(X(t_k)), \mu_T(t_k))) \quad (9)$$

4.2 Multi-agent Robotic Simulation

4.2.1 Architecture

The V-Lab® architecture was successfully used to demonstrate simulation of multiple robots. Each of the modules, shown in figure 8, such as rover dynamics, sensors, terrain, etc were implemented as DEVS atomic or coupled models. The controller for robots was tested and implemented using crisp and fuzzy logic separately. Since the V-Lab® architecture provides plug-and-play, the controller can be easily removed and replaced without affecting functionality of other modules like terrain, plot or rover dynamics.

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.

4.2.2 Simulation Results

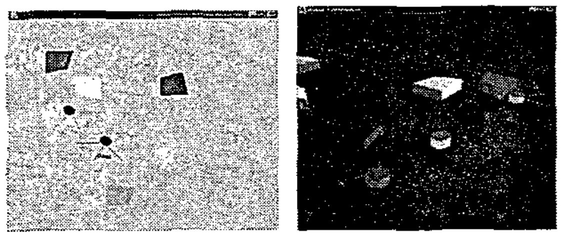

Figure 14. Robotic Simulation using V-Lab®

Figure 14 shows the result of robotic simulation of rovers reaching a goal position avoiding the obstacles (convex polygons). Several rovers can be added to create swarms of robots performing collaborative task. Simulation can be performed on different machines using *Java Sockets*.

5 Conclusion

In this paper we described multi-agent network security simulation using temporal fuzzy logic implemented within IDEVS framework. For complex multi-agent simulation we defined a modular and distributable framework called Virtual Laboratory (V-Lab®) which is built on top of IDEVS components. The control algorithms simulated using IDEVS components will be tested on hardware platform using *ActivMedia* Robotics Pioneer II robots. As a future work, GA-DEVS can be used to optimize the temporal Fuzzy-DEVS rules.

6 References

[1] Arizona Center for Integrative Modeling and Simulation, http://www.acims.arizona.edu/.

[2] 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

[3] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., "Design Patterns: Elements of Reusable Object- Oriented Software", Addison Wesley Publication, Oct 1994.

[4] Jamshidi, M., L. dos S. Coelho, R. A. Krohling, and P. Fleming, "Robust Control Design Using Genetic Algorithms", *CRC Publishers, Boca Raton, FL, 2002.*

[5] Jamshidi, M., Zilouchian, A., "Intelligent Control Systems using Soft Computing Methodologies", *CRC Press, Boca Raton, FL, 2001*

[6] Mucientes, M., R. Iglesias, C.V. Regueiro, A. Bugarin, P. Carinena, S. Barro, "Fuzzy Temporal Rules for Mobile Robot Guidance in Dynamic Environments", IEEE Transactions on Systems, Man, and Cybernetic – Part C, August 2001.

[7] Packet Filtering HOWTO, http://www.netfilter.org/

[8] US Department of Defense, "High Level Architecture", https://www.dmso.mil/public/transition/hla/

[9] Wollrath, A. and Waldo, J. "Trail : RMI", http://java.sun.com/docs/books/tutorial/rmi/index.html

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

[11] 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*