V-Lab—A Virtual Laboratory for Autonomous Agents—SLA-Based Learning Controllers

Aly I. El-Osery, *Member, IEEE,* John Burge, Mo Jamshidi, *Fellow, IEEE,* Antony Saba, Madjid Fathi, *Senior Member, IEEE,* and Mohammad-R. Akbarzadeh-T., *Senior Member, IEEE*

***Abstract*—In this paper, we present the use of stochastic learning automata (SLA) in mutliagent robotics. In order to fully utilize and implement learning control algorithms in the control of multiagent robotics, an environment for simulation has to be first created. A virtual laboratory for simulation of autonomous agents, called V-Lab is described. The V-Lab architecture can incorporate various models of the environment as well as the agent being trained. A case study to demonstrate the use of SLA is presented.**

***Index Terms*—Reinforcement learning, robotic agents stochastic learning automata (SLA), V-Lab.**

I. Introduction

THE PAST century has seen an evolution in the way man envisions robotics, from the earlier mechanical devices performing purely repetitive tasks to their more recent computer controlled and more intelligent counterparts. Even though several issues in robotics such as kinematics, dynamics, and control of manipulator arms in known environment seem to have reached a relative level of maturity, several new issues and applications have risen in the past several years which deal with increased need for autonomy and intelligence of robots, increased uncertainty in robot environments, and increased complexity in coordinating robot-to-robot interaction and cooperation.

Applications of the multiagent architecture are many. Some of the earlier applications of multiagent collaboration and coordination were in part-assembly where two or more robots worked to assemble two or more pieces of hardware. Human-like robot hands were an example of several manipulator arms (fingers) cooperating in order to handle an ill-defined object. More recent applications have included self-healing minefields and formation flying of multiple air vehicles or satellites. For migrating birds, the formation flying provides for decreased overall drag, and hence, increased range. In the case of air vehicles and microsatellites, we can also expect increased flexibility and robustness at lower cost [1], and increased efficiency in interferometeric synthetic aperture radar (InSAR) [2].

Furthermore, the advantages of having multiple agents are not limited to tasks too large for a single agent alone. Multiple cooperating agents also promise increased mission robustness and learning ability. Using multiple agents also allows for the failure of a single agent without the failure of the entire mission. Other researchers have commented that the capability of n cooperating robots is higher than the sum of their individual capabilities. Specifically, Berenji and Vengera [3] showed the learning capability of n cooperating agents is higher within the reinforcement-learning paradigm. This point can perhaps be demonstrated by considering the success of social animals such as humans, bees, and ants. Human societies are an elaborate example of a large number of task-specific multiple agents who cooperate in order to achieve a higher standard of success than any one human being could obtain acting alone.

This promise of added capability and enhanced performance is, however, provided at additional cost of added system complexity. Here, we introduce "social intelligence" as a framework of multiagent cooperation. In 1997, Dudek *et al.* [4] described several axioms for multiagent robotics which differentiated among various approaches based on collective size and reconfigurability, communication range and composition, and the processing ability of each agent within the collective. Fig. 1 illustrates a different breakdown where various ways of handling multiagent robotics can be categorized by their framework of intelligence, architecture of cooperation, learning and optimization, type and level of communication and security, world environment and task type, and their architecture of perception.

Learning is a common aspect of multiagent robotics where robustness and performance is demanded in the face of environmental uncertainty. Various algorithms may be used in the learning that are often closely interrelated with the collective's cooperation architecture. Uchibe *et al.* [5] used genetic programming for coevolution of three mobile robots in a soccer game where both cooperation as well as competition behaviors were obtained. Nilli-Ahmadabadi [6] investigated nonhomogeneity in a different light, where agents had similar mechanical capabilities, but differing levels of expertise. Applying reinforcement type learning, they showed that cooperative learning is accelerated when agents have different experiences.

Manuscript received November 15, 2001; revised March 1, 2002. This work was supported by NASA Grant NAG2-1547. This paper was recommended by Guest Editors M. S. Obaidat, G. I. Papadimitriou, and A. S. Pomportsis.

A. I. El-Osery was with the Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87106 USA. He is now with the Department of Electrical Engineering, New Mexico Institute of Mining and Technology, Socorro, NM 87801 USA (e-mail: aly.elosery@ieee.org).

J. Burge and A. Saba are with the Department of Computer Science, University of New Mexico, Albuquerque, NM 87106 USA.

M. Jamshidi and M. Fathi are with the Autonomous Control Engineering Center, Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87106 USA.

M.-R. Akbarzadeh-T. is with Ferdowsi University of Mashhad, Mashhad 91368, Iran.

Publisher Item Identifier S 1083-4419(02)06461-0.

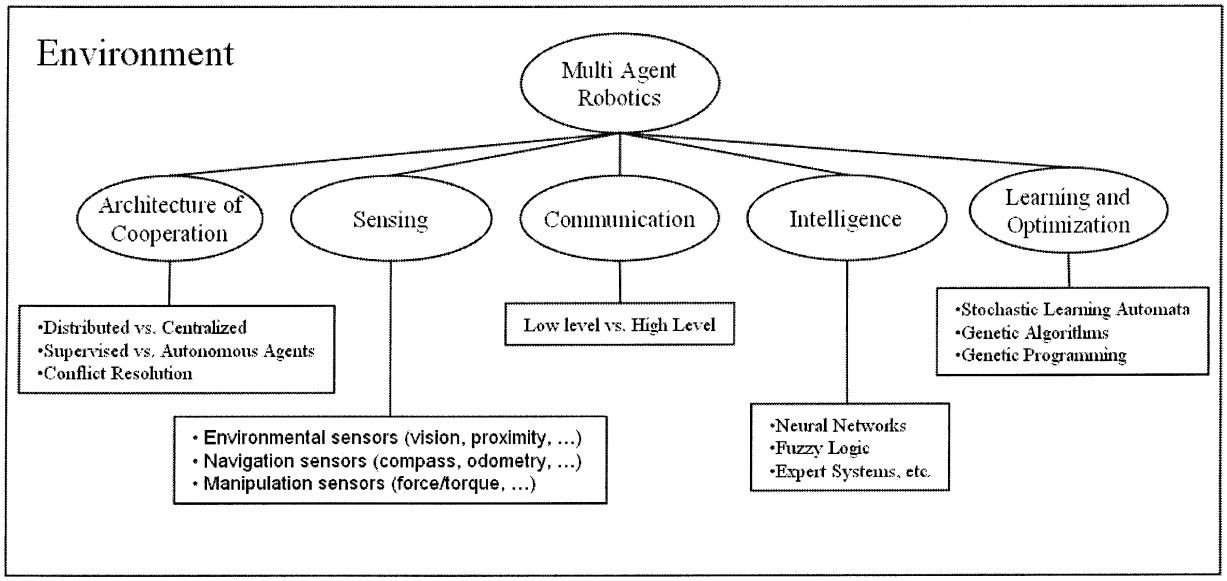

Fig. 1. Multiagent robotics is also a multifaceted field.

Fu [7] is probably the first to introduce learning control. Based on Fu's definition, a learning controller is one that learns the information during the operation and the learned information is, in turn, used as an experience for future decisions or controls. Several researchers have investigated different types of learning control algorithms (e.g., Chandrasekharan and Shen [8], and Lakshmivarahan [9]) and their applications to real-life problems such as active vehicle suspension [10], path planning for manipulators [11], and path control in an automated highway systems [12]. Jamshidi *et al.* [13], [14] have proposed a learning control approach for two dimensional systems with applications in robotic manipulators and nuclear reactors. In this paper we discuss the use of learning control for multiagent robotics using a *stochastic learning automata (SLA)*.

In order to implement SLA, first a simulation environment, called V-Lab, is being created to facilitate testing the applications of SLA. One of the most powerful outcomes of the digital computer has been its role in representing real physical environments through modeling and simulation. A *model* is a representation of a real system. Often, physical principles are the cornerstone for creating mathematical models, be it lumped parameter, distributed parameter systems, or heuristic models. Simulation utilizes a model (mathematical or otherwise) to make a hypothetical realization of a real physical system and allows the user to simulate numerous alternatives based on "what if" situations. In many circumstances, several physical models and simulations need to interconnect and take place simultaneously for a larger more complex simulation to yield results. This requirement leads to the notion of *distributed modeling and simulation* in a multiphysics environment.

In this paper, we intend to cover three complementary paradigms, namely, the SLA as a learning paradigm; V-Lab as a framework of interaction and modeling, which includes discrete-event system specifications (DEVS) [15]–[17]; and multiagent robotics as an application. The multiteacher penalty/reward within the SLA learning paradigm allows for multiaspect multiobjective learning. The cooperative architecture is centralized where a more intelligent master robot plans the execution of complex tasks, divides them into simple tasks and accordingly instructs the slave robots to carry on simple tasks. Furthermore, stochastic learning is embedded in all levels of this discrete-event-based hierarchy. The resulting multiagent system is analogous to an army battalion where soldiers carry on simple tasks as directed to them; yet, the resulting action of the collective can be quite complex. The following approach is unique in that it applies stochastic learning paradigm to DEVS representation of a mobile robot collective. It is felt that this view of multiagent robotics can have strong applied implications in both the realms of stochastic learning as well as multiagent robotics.

The paper is structured as follows. Section II introduces the various desirable features of the proposed V-Lab simulation environment. Section III briefly introduces DEVS as the collective's general framework of cooperation and communication. In Section IV, the V-Lab structure is discussed. An SLA is discussed in Section V. Finally, conclusions and future works are presented in Section VI.

II. V-Lab Overview

The design of distributed simulations frequently becomes large and complex. Applying a layered pattern 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. In this section, we introduce a definition of a layered framework that allows for the construction of distributed-agent-based simulations in a virtual laboratory, V-Lab.

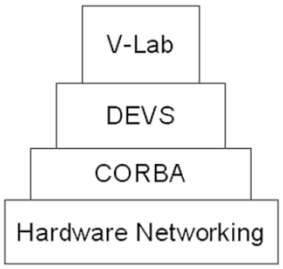

Fig. 2. Distributing simulation layers using DEVS.

A. V-Lab Environment

The V-Lab environment consists of four distinct software layers, as shown in Fig. 2, 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, the common object request broker architecture (CORBA) [18] 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 DEVS environment is introduced in Section III. 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 in which they communicate. It also provides the critical objects needed to control the flow of time, the flow of messages, and the base classes objects designers will need to create their own V-Lab modules. The design of V-Lab is introduced in Section IV.

III. Discrete-Event Systems

The DEVS environment is a distributed modeling environment developed by the modeling and simulations group at the University of Arizona headed by Zeigler [16], [17]. It was created to provide a robust and nonspecific environment for modeling and simulation projects. The DEVS environment expands the capabilities of more generic distributed architectures such as CORBA. The DEVS environment provides 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.

Layered on top of the DEVS environment are the models that a developer would create to compose a simulation. These models are divided into two categories: 1) *atomic* and 2) *coupled*. Atomic models compose the functionality of the basic units in a simulation. Using these atomic models as building blocks, coupled models build up the simulation by linking them together. Thus, simulations using DEVS are collections of models composed in a hierarchical fashion. For instance, a DEVS coupled model ABC, such as the one in Fig. 3, can be constructed from an atomic and a coupled model, A and BC, respectively. BC is itself a coupled model that is constructed from two atomic models, B and C.

The hierarchical specification defines which models are included as submodels for any given coupled model, but it does not define how these submodels interconnect with the parent model or with each other. This information is defined in the form of ports and couplings. Fig. 4 illustrates one such coupling for the ABC model.

In the example, the input into ABC is coupled with the input in A. In effect, this transfers all messages coming into the ABC model on its in-port to the in-port on the atomic model A. The output of A is then coupled to the input of the BC model and the output for BC is coupled to the output of ABC. A similar set of couplings is defined for the coupled model BC. If the ABC model itself was coupled to other models, messages passing to the ABC model's output ports would be passed to the input ports of these models.

In order to fully define an atomic model, the following information must be specified:

1) the input ports that receive external events;
2) the output ports that send external events;
3) generally, two state variables, *phase* and *sigma*, indicating what state the model is in, and how long it will be in that state;
4) a time advance function that controls the timing of the internal transition functions (frequently based on *sigma*);
5) an internal transition function that determines which state the model will go to after being in state *phase* for the duration indicated by *sigma*;
6) an external transition function that determines which state the model will go into from state *phase* after receiving a message on an input port and how long it will stay in that state;
7) a confluent transition function that determines the order in which the internal and external transition functions will occur;
8) an output function that generates external events just before an internal state transition occurs.

In order to fully define a coupled model, the following information must be specified:

1) the models from which the coupled model is composed;
2) the input ports that receive external events;
3) the output ports that send external events;
4) the coupling specification that ties the input and output ports of the coupled model to input and output ports of the models contained within the coupled model;
5) the coupling specification that ties the input and output ports of the models contained within the coupled model together.

The interaction between all of the models, both coupled and atomic, comprises a simulation. Fig. 5 illustrates the order of events that generally takes place in the execution of a simulation. First, the models are created. The coupling between the models is then set up. After that, all the models are initialized and then a message is sent to the highest coupled model that starts the DEVS simulation cycle. The simulation loops through

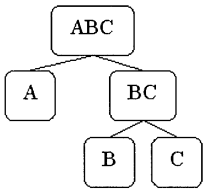

Fig. 3. Hierarchal tree for model ABC.

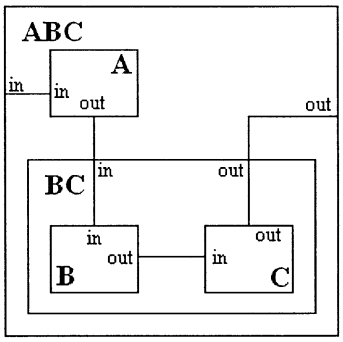

Fig. 4. Coupling relation for model ABC.

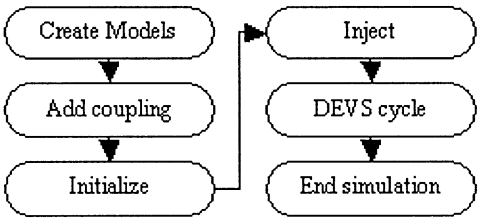

Fig. 5. Control flow of a simulation using DEVS.

the DEVS cycle until the termination conditions are met, at which point the simulation will end.

At each pass through the DEVS cycle, a message is passed down the hierarchical tree of models to determine which atomic models are *imminent*. Imminent models are models that are scheduled to perform an internal transition at the current time step. If no model is imminent, the pass through the DEVS cycle will be over. (Note that the DEVS cycle is slightly more complex than this and is able to skip cycles in which no models are imminent.) If there are imminent models, they will generate external events on their output ports and proceed through their internal transition.

After the imminent models have completed their transitions, all the external events they created are sent to the models that they are coupled to. Each of these models will then call their external transition function to change their states. The completion of this step ends a single iteration of the DEVS cycle.

In addition to providing classes for models and the environment in which simulations run, the DEVS environment provides an abstract data class library. Fig. 6 lists these classes and their hierarchical structure. This library provides classes which can be used to store, retrieve, and organize objects used in the simulation [16].

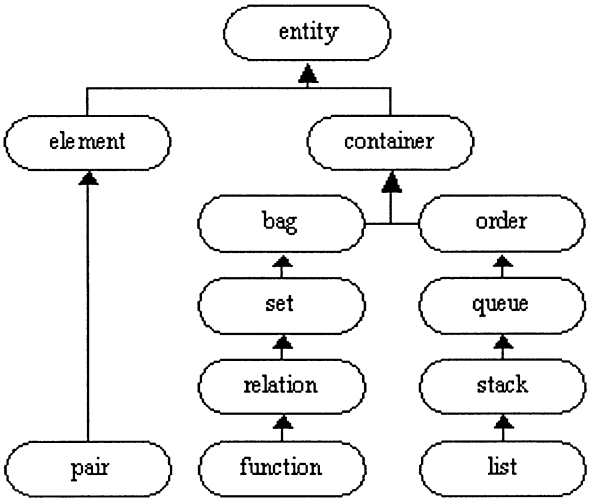

Fig. 6. Class hierarchy for containers.

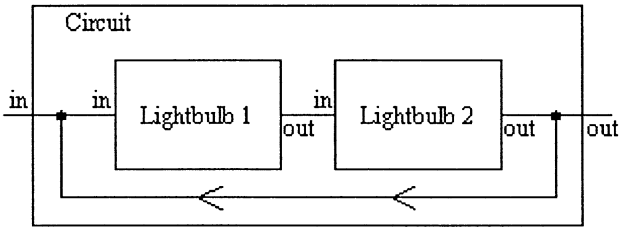

Fig. 7. Coupling diagram for example simulation.

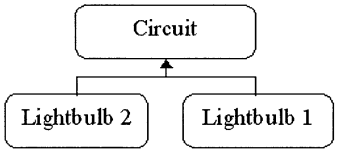

Fig. 8. Hierarchical tree for example simulation.

A. An Example: Simple Light-Switch Model

There is no better way to illustrate the functionality of the DEVS environment than by illustration of a simple example. The following simulation models two switches that each controls a light bulb. When a switch is pressed, its respective light bulb will turn on. Fifteen seconds afterwards, the light bulb switches off and flips on the switch for the other light bulb. Similarly, when the second light goes off, it causes the first switch to turn on which causes a cycle that will then repeat endlessly.

Fig. 7 shows the coupling diagram for the model, and Fig. 8 gives the hierarchical construction of the model. The simulation is constructed from a single coupled model, Circuit, which is constructed from two instances of a single atomic model, Lightbulb. The Circuit class defines the coupled model, and the two atomic models are defined from the Lightbulb class. The simulation will follow the general order of events that take place in most DEVS simulations (see Fig. 5). First, the coupled model Circuit will be created. As its constructor is called, it will create both of the atomic models Lightbulb 1 and Lightbulb 2. It will then couple all the models together. Each model initializes itself

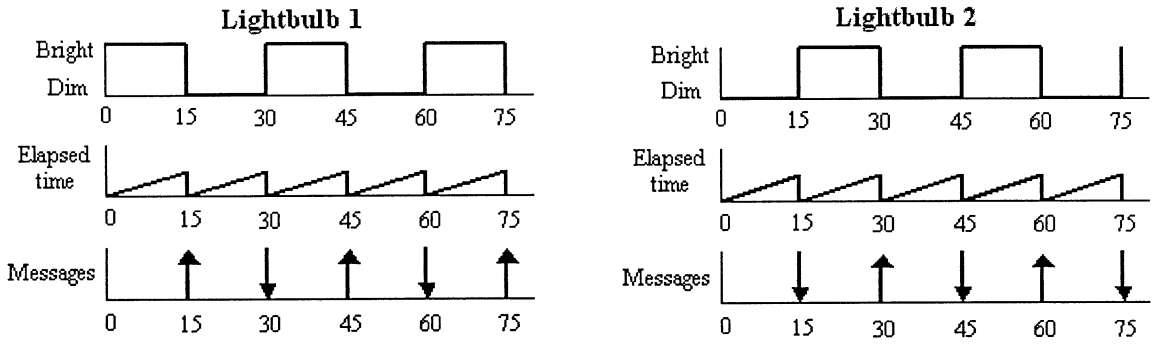

Fig. 9. State and timing diagrams for example solution. ↑ indicates the model is sending out a message. ↓ indicates the model is receiving an incoming message.

and the simulation will be ready to run. At this point, a message will be injected into the Circuit instance and a single round of the DEVS cycle will execute. After each cycle, another message will be injected into the Circuit instance and the DEVS cycle will repeat indefinitely.

Fig. 9 contains three diagrams for each of the atomic models as the simulation runs through time. The first is a state diagram that shows the changing values of the *phase* variable. Notice that Lightbulb 1 and Lightbulb 2 are constantly in different states. The second diagram shows the elapsed time each model has been in its current state, and the last diagram shows where there is an incoming and outgoing message. Incoming messages are indicated with a "down" arrow, and outgoing messages are indicated with an "up" arrow. For every message in one atomic model, there is an equivalent opposite message in the other atomic model.

In the first iteration of the DEVS cycle, Lightbulb 1 is in the *bright* state with a *sigma* value of 15, and Lightbulb 2 is in the *dim* state with a *sigma* value of infinity. After 15 iterations of the DEVS cycle, the internal transition function of Lightbulb 1 is called which causes it to go from the *bright* state to the *dim* state. However, just before the internal transition occurs, Lightbulb 1's output function is called, sending a message to the input port to Lightbulb 2. In turn, this causes Lightbulb 2's external transition function to be called causing Lightbulb 2 to enter the *bright* state with a *sigma* value of 15. After 15 more DEV cycles, an equivalent sequence event occurs between Lightbulb 2 and Lightbulb 1.

IV. Proposed V-Lab Architecture

V-Lab models can be categorized into one of six groups:

1) *SimEnv* and *SimMan;*
2) *control models;*
3) *agent models;*
4) *physics models;*
5) *terrain models;*
6) *dynamic models.*

Each of these models is referred to as a *high-level model* and is constructed from atomic and coupled models that are not high-level models.

Figs. 10 and 11 give the coupling diagram for the relation of all the high-level models and the hierarchical layout, respectively. The heart of V-Lab, and the first type of high-level models, are the SimEnv coupled model and the SimMan atomic model, which are common to every V-Lab simulation. SimEnv is the highest level coupled model in the simulation, and is responsible for creating the instances of all of the other high-level models. It acts as the housing for all the high-level models in the simulation. SimMan is an atomic model to which all other models in SimEnv connect. It is responsible for coordinating messages between other high-level models, controlling the flow of time in the simulation, and tracking information about the state of the agents in the simulation. All of the high-level models have at least one input port and one output port tied to SimMan.

The second type of high-level models are the control models. They store the behavior algorithms that will be used to control the agent, physics, and terrain models. The control objects can be based on fuzzy logic, SLA, neural networks, etc. Agent control models indirectly determine what an agent will do by controlling the agent's actuators, which will later cause a dynamic model to change the state of an agent.

The third type of high-level models are the agent models which contain sensor models and actuator models. The sensor models contain the information about the environment that the agent is aware of and the actuators contain the information that dynamic models use to modify the agent and the environment.

The fourth type of high-level models are the physics models. These models are used to model the world physical phenomena present in the simulation. How the agent interacts with the environment and how agent sensors set their state is encapsulated within the physics models.

The fifth type of high-level models are the terrain models. The terrain models contain information about the layout of the simulation's environment and are analogous to maps.

The sixth, and last, type of high-level models are the dynamic models. These models are responsible for making changes based on the state of the agents and their actuators to the agent models and to the information SimMan tracks about the agent models.

Since each of the high-level models is coupled to SimMan, when one high-level model needs to get information from another high-level model it can (and must) do this through

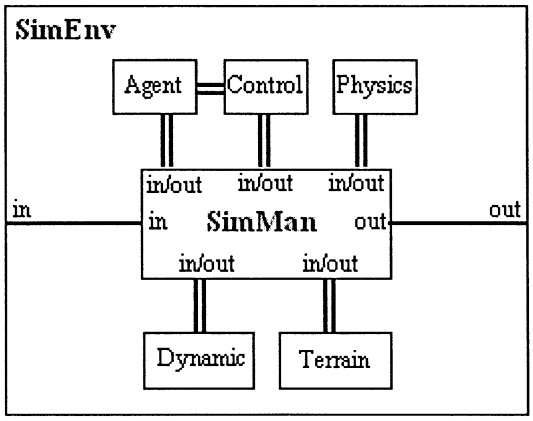

Fig. 10. Relationship diagram for high-level models.

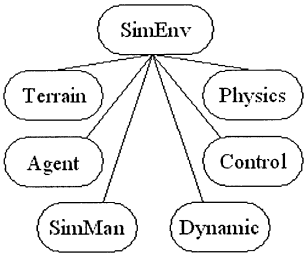

Fig. 11. Hierarchical model structure for SimEnv and SimMan.

SimMan.[1] This increases the amount of time it takes to send messages from one model to another, but it greatly reduces the amount of coupling required between any two arbitrary models.

Fig. 12 shows the port connections between all of the high-level models, and the typical models that they contain. SimMan has a specific set of ports coupled to every high-level model. SimMan contains one output port to the control algorithm in every control model, one output port for each of the sensors in the agent model, one input port from the agent's actuators, and one set of input–output ports tied to the physics, terrain, and dynamic models.

SimMan also controls the timing and sequence of events that occur in the simulation. Fig. 13 illustrates the flow of time, controlled by SimMan, in a typical simulation. Once SimMan gets a signal from SimEnv that initiates the DEVS cycle, SimMan will perform iterations of the V-Lab cycle until the simulation terminates.

A. V-Lab Cycle

Phase 1 marks the start of the V-Lab cycle in which SimMan checks to see if the termination events are satisfied. If they are, the cycle stops. If they are not, SimMan enters Phase 2 of the V-Lab cycle.

At the beginning of Phase 2, the state of the agent's sensors needs to be updated. SimMan sends a message to each agent's sensor. Once the sensors receive this message, they will send out requests for their state information back to SimMan. These messages will be routed to any physics models that can fill in the required information. The physics models may send

[1]With the exception of control models and the agent models they control.

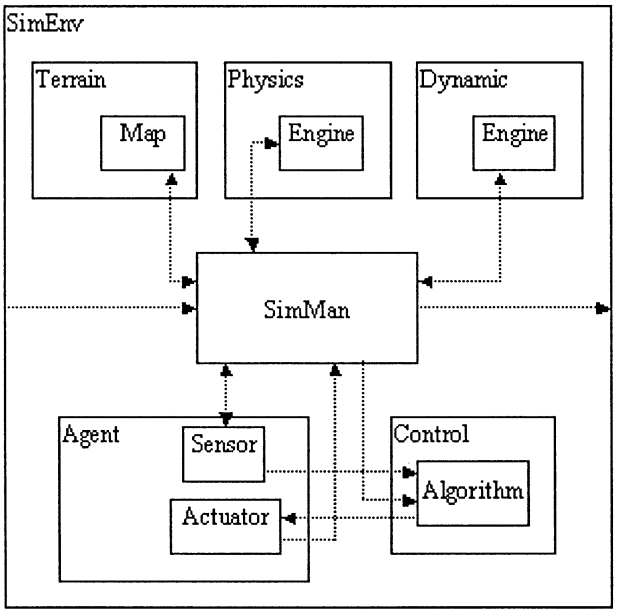

Fig. 12. Coupling diagram example for SimMan.

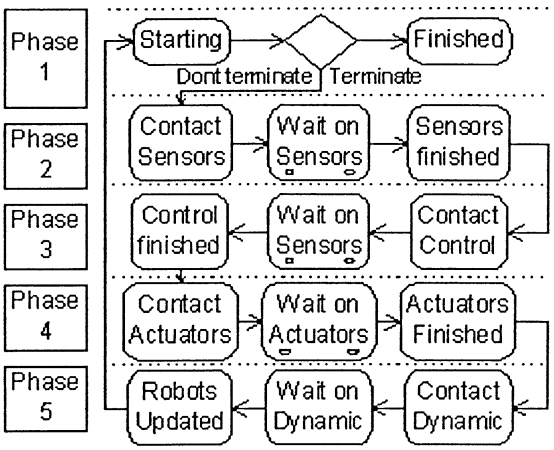

Fig. 13. Phase and state diagram for SimMan.

more messages back to SimMan requiring information from other high-level models. After the physics models have finished their calculations they will send their results back to SimMan. SimMan then forward them back to originating sensor. At this point, the sensor will send a message back to SimMan indicating that it has finished. They will also send out a message to the control algorithms with their new state. Once all the sensors have finished, SimMan proceeds into Phase 3.

Phases 3, 4, and 5 contain similar parlays of messages between SimMan and models. However, in these phases, the primary models with which SimMan communicates are the control, actuator, and dynamic models, respectively.

During all of the phases of the V-Lab cycle, many messages are being passed between the myriad of DEVS models that comprise the simulation. It is important for the objects that represent the messages to be suitable for a wide range of possible messages that can exist. V-Lab defines a Message class that acts as a base class for any object that acts as a message sent between the high-level models.

Fig. 14 visualizes the structure of a typical message object. The message is broken up into two sections, the header section and the data section. The header information consists of information that is used by SimMan to track and route the message to and from the sender and all valid receivers. This section contains information such as the name of the originating high-level model, the message identification number, whether or not the message represents a request for information or the supply of information, etc. Message classes that inherit from the Message base class will generally not modify the information that is assembled in this section.

The data section represents the actual information that is being passed between two high-level models; however, certain messages do not contain a data section. Typically, Message classes that inherit from the V-Lab Message class will add data structures to this section of the message.

B. Multiprocess V-Lab

In order implement the V-Lab architecture on multiple platforms, each platform will contain a single SimEnv model. That SimEnv model will be coupled to each of the other SimEnv models on other machines. On a solitary machine, a SimMan module routes all the messages from its high-level models to other high-level models with which it is connected. If that SimMan module cannot find another high-level model that can respond to the message, it will send that message out to the other SimMan objects on different machines. Each of those SimMan models will respond with either the correct message response or an indication that it does not contain a high-level model that can respond to the message.

Excessive message passing between models in the same process should always be avoided, but in the case of inter-machine message passing, excessive message passing can bring the simulation execution to a grinding halt. It is important to keep high-level models that are highly dependent on each other located in the same machine. This is not a trivial problem and future work needs to be done to optimize this process in the general case.

C. Requirements for a Specific Simulation

V-Lab provides the framework for a simulation designer to begin his work. However, in order to fully create a simulation, the designer will need to develop the actual components of the simulation (or reuse existing components). For a given simulation, the simulation designer is responsible for providing the following:

1) a class named *SimContents*;
2) all of the high-level models;
3) message classes;
4) a termination function.

SimContents defines which high-level models will be included in the simulation. The designer provides these high-level models; however, the designer will not have to decide how the models are coupled together as this is primarily defined by the V-Lab architecture. The designer will also have to provide subclasses of the Message class, which will define the structure

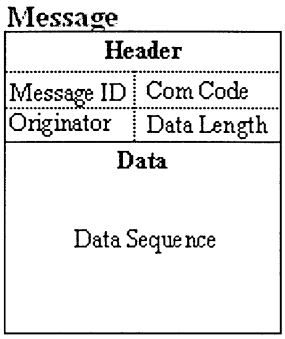

Fig. 14. Typical message object.

of the messages being passed between the high-level models. Finally, in order for SimMan to know when the simulation will end, the designer will have to provide a Boolean termination function that returns true if the simulation has met its termination conditions, and false otherwise.

Once the designer has created all of these components and connected them together with the V-Lab architecture, the simulation can be run. Furthermore, after all of these components have been created, it is a relatively easy task to modify the components of the simulation in order to run a wide variety of test runs in the same or similar environments. The modularity of the components in V-Lab adds a tremendous amount of power and flexibility to the simulations run on its architecture.

V. Stochastic Learning Automata

Control theory, including optimal control theory, often requires perfect information or *a priori* information of the system to be controlled. In many practical problems, this information may not be available. Hence, in these situations, as an alternative approach, the use of learning control algorithms becomes necessary. Learning control is of particular interest in distributed robotics. In most cases, the robots are sent into an environment of which little knowledge is available, i.e., contaminated depository or Mars surface. The robots are then required to learn their environment and perform their task with minimal error. An important tool in learning control is SLA. The basic operation of SLA is as follows. At any given time, an action is performed by the SLA based on its internal states. Due to that action, the environment (also referred to as a "teacher") responds with a value between zero and one. A value of zero corresponds to full reward and a value of one corresponds to full penalty. Based on this feedback from the environment, SLA updates the probabilities of choosing a certain state. This process is repeated until the average penalty is minimized.

Learning automaton (LA) needs no knowledge of the model of the process to be controlled or any analytical knowledge of the function to be optimized. It is connected in feedback loop to the environment (see Fig. 15). LA is a sequential machine characterized by a set of:

1) internal states;
2) input actions;
3) state probability distributions;
4) reinforcement scheme;
5) output function.

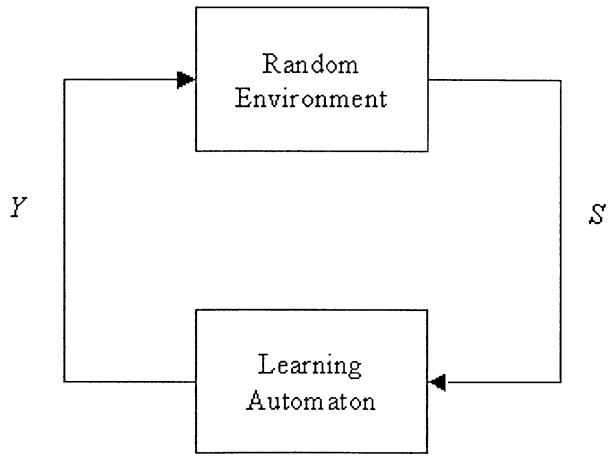

Fig. 15. Feedback loop of a learning automaton.

The probability distribution of the actions is adjusted using the reinforcement scheme to achieve the desired objective.

A. Terms and Definitions

In this section, some of the variables and terms used in SLA are introduced. A stochastic automaton is a quintuple $\{S, W, Y, F, G\}$ [7]. S is a finite set of inputs, $S = \{s^1, \ldots, s^q\}$, W is a set of finite internal states $W = \{w^1, \ldots, w^r\}$, Y is finite set of outputs $Y = \{y^1, \ldots, y^m\}$, F is the next state function

$$w(n+1) = F\left[s(n), w(n)\right] \tag{1}$$

and G is the output function

$$y(n) = G\left[w(n)\right]. \tag{2}$$

The probability of choosing a certain state or action, w^i, is $p_i(n)$. The probability of a penalty is denoted by c^i, and hence, the probability of a reward is $1 - c^i$. Fig. 16 shows the operation of an SLA block.

B. Basic Operation of an SLA

If the state w^i is chosen at time step n, then the stochastic automaton performs action y^i. As a result of this action, a reward $(s = 0)$ or a penalty $(s = 1)$ is assigned. The SLA then updates the probabilities of the internal states.

The performance of the SLA is determined using the mathematical expectation of penalty [7]

$$I = \lim_{n\to\infty} (1/n) \sum_{i=1}^{n} s(j). \tag{3}$$

If

$$I < (1/q) \sum_{j=1}^{q} c^j. \tag{4}$$

The performance of the SLA is called *expedient*. Expediency reflects the closeness of I to $I_{\min} = \min(c^1, \ldots, c^q)$. If $I = I_{\min}$, then the SLA is said to be optimal. The idea is to update the probability distribution in order to achieve expedient performance. This may be accomplished using a reinforcement scheme.

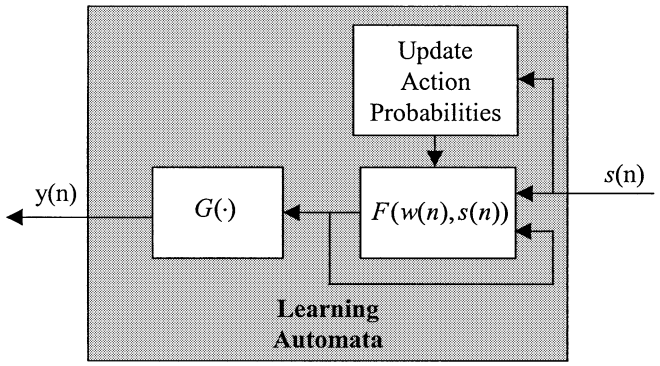

Fig. 16. SLA block.

C. Reinforcement Learning

In learning control, the basic problem is to determine the control action y^* such that

$$E\{\xi|s, y^*\} = \min_k \left\{E\left[\xi|s, y^k\right]\right\} \tag{5}$$

where $\xi = f(s, y^i, s')$ is the instantaneous performance evaluation of the action y^i following reaction s, and s' is the response due to the action y^i after a reaction s. In this section, Fu's reinforcement learning algorithm is presented [7]. Let $\lambda_i(n)$ be the variable relating the system performance measure to the response of the stochastic environment due to the action y^i of the LA. Then, the linear reinforcement algorithm may be of the form

$$p_i(n+1) = \alpha p_i(n) + (1-\alpha)\lambda_i(n+1) \tag{6}$$

$$\text{for } i = 1, \ldots, q \qquad \text{where } 0 < \alpha < 1$$

and

$$\lambda_i(n+1) = \begin{cases} 1, & \text{if } \hat{E}_{ni}\left\{\xi|s, y^k\right\} = \min_k \hat{E}_{nk}\left\{\xi|s, y^k\right\} \\ 0, & \text{if } \hat{E}_{ni}\left\{\xi|s, y^k\right\} \neq \min_k \hat{E}_{nk}\left\{\xi|s, y^k\right\}. \end{cases} \tag{7}$$

$\hat{E}_{ni}\{\xi|s, y^k\}$ is the stochastic approximation used to estimate $E_{ni}\{\xi|s, y^k\}$.

According to Fu [7], if we let

$$p\left\{\hat{E}_{ni}\left\{\xi|s, y^k\right\} = \min_k \hat{E}_{nk}\left\{\xi|s, y^k\right\}\right\} = \varphi_j(n+1) \tag{8}$$

then

$$E\left[p_j(n+1)|p_j(n)\right] = p_j(n) + (1-\alpha)\varphi_i(n+1). \tag{9}$$

If

$$P\left\{\lim_{n\to\infty} \varphi_j(n) = 0\right\} = 1, \qquad \text{for every } y^j \neq y^*, \tag{10}$$

$$p_j(0) > 0, \qquad \sum_{j=1}^{m} p_j(0) = 1 \tag{11}$$

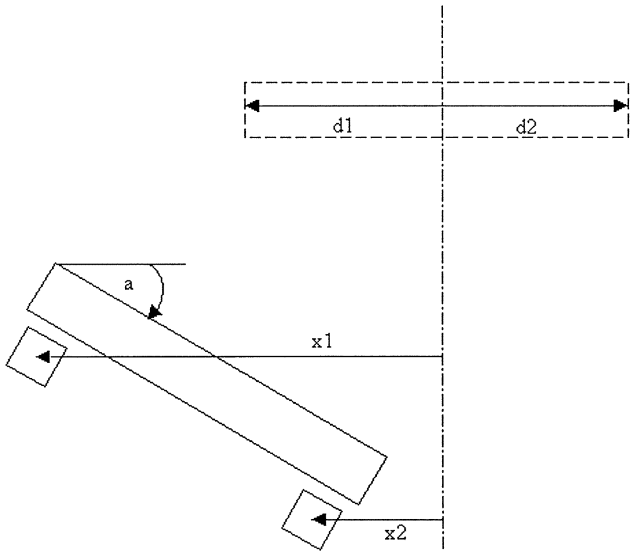

Fig. 17. Top view of the simulation.

and

$$p_i(n) = \max_{s^k} \{p_k(n)\}, \qquad i, k = 1, \ldots, m \quad (12)$$

then

$$P\left\{\lim_{n\to\infty} P\left[y^* | y(n)\right] = 1\right\} = 1 \quad (13)$$

which means that the desired optimal control law will eventually be reached with probability one [19].

D. Simulation

In this section, the case study used to test the SLA is presented. In this case study, two robots are to collaborate together to perform a certain task. It is assumed that all the measurements available to the robots are corrupted with white Gaussian noise. Assume that we have an object that needs two robots to move it and push it to a preassigned location. The robots have no information about the size of the object, the environment they are in, or the location to which the object is to be taken. The robots also have no information about the function that needs to be optimized, making the problem more complex. These two robots could be thought of as robotic agents on a distant planet, and their mission is to collect rock samples from that planet. At times, the rock samples may be too large for one robot to handle, and hence, the need for multiple robots to collaborate becomes necessary. Due to lack of the information about the environment, the models of the robot, or the function to be optimized, it may not be possible to use traditional control methodologies; instead we use SLA to determine the actions of the robots.

The two robots communicate with a central station that responds to the action of the robots by either a reward or a penalty. Each robot is equipped with simple sensors to allow them to locate the object and to align themselves. The central station has no information about the actual mathematical model of the robots and the data it collects is corrupted by noise. It is assumed that the robots have a mechanism for locating themselves at the opposite ends of the object. Using the software environment Webots [20], the simulation was carried out. The simulation includes the physical model of the robots, the object, and the environment.

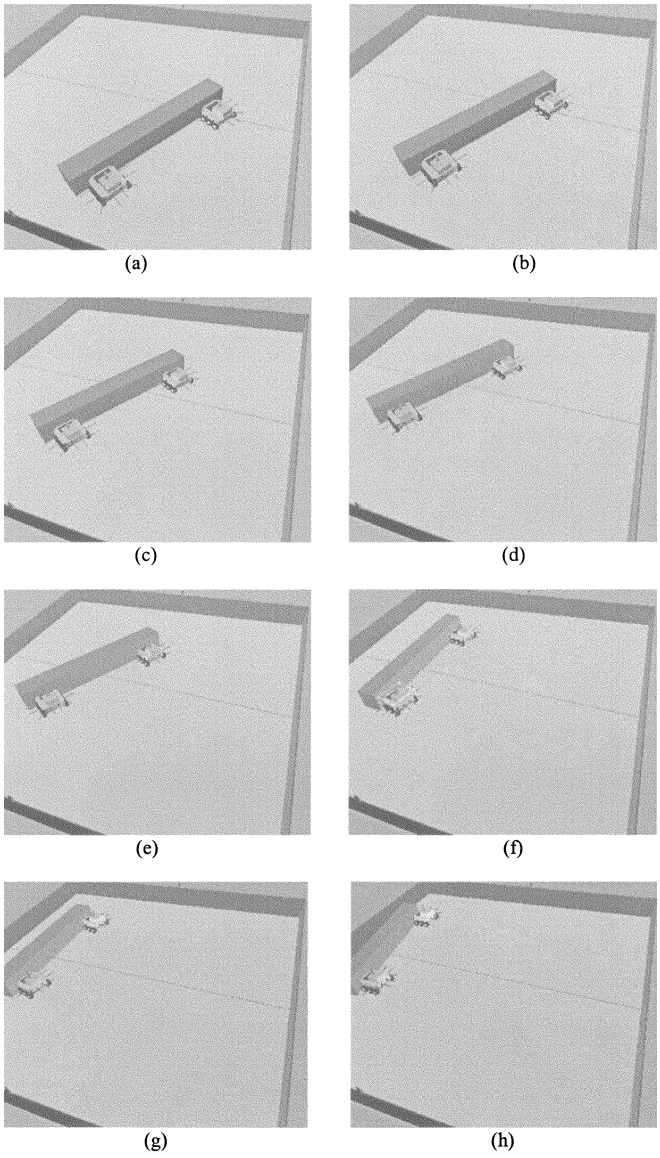

Fig. 18. Simulation results.

The goal of the SLA is to determine the set of actions needed by the two robots in order to orient the object and push it to the desired location. The possible actions that can be taken by the robots are:

1) both robots push forward;
2) left robot pushes;
3) right robot pushes.

As the simulation progresses, the central station sends a penalty or a reward, based on the actions of the robots. If the robots perform the wrong sequence of actions, the block will not be delivered to the desired location. The penalty or the reward is determined with the help of Fig. 17. If $x1$ is greater than $d1$, then pushing forward or having only the left robot push will be rewarded. If angle a is greater than 60°, then

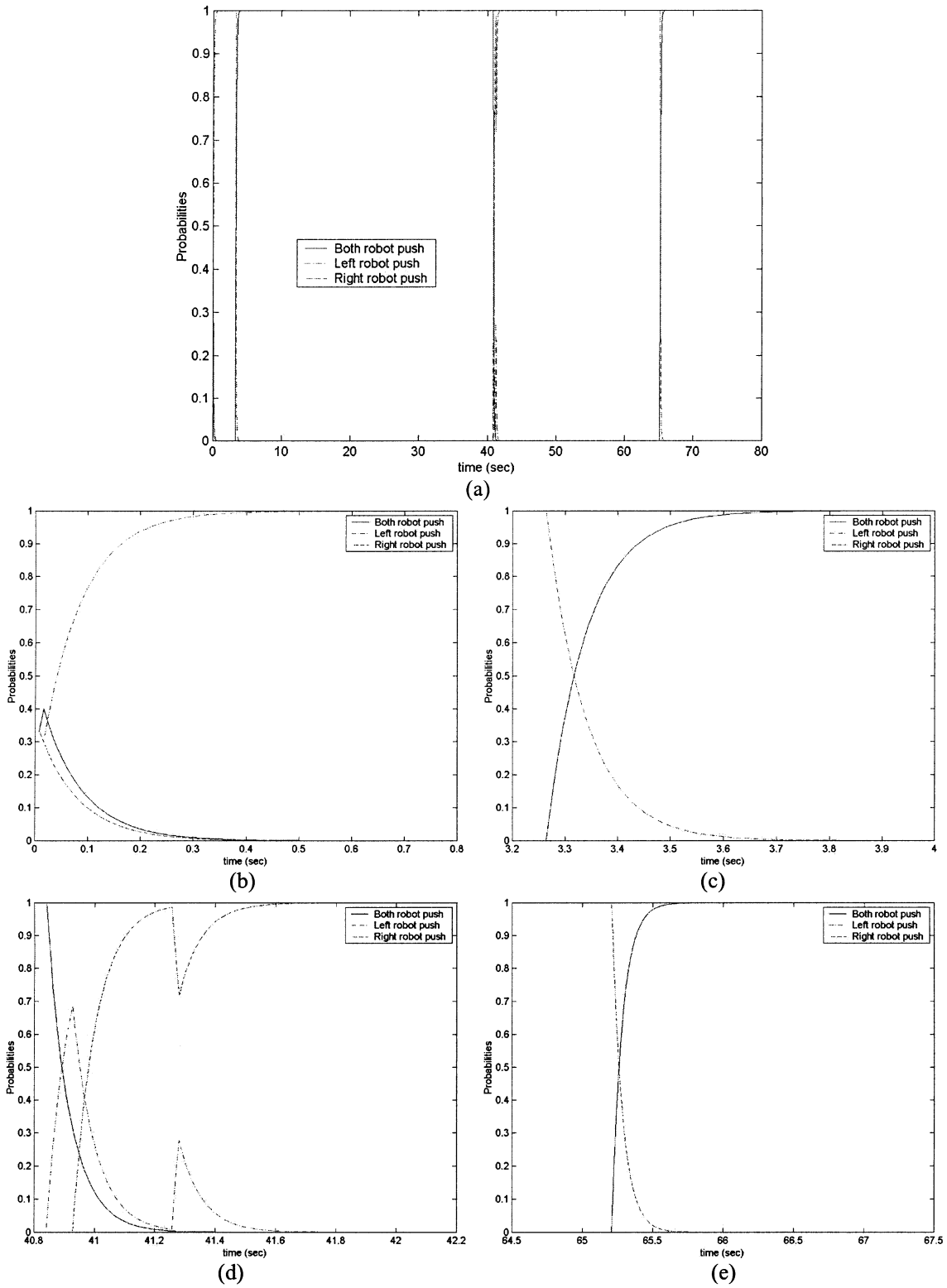

Fig. 19. Plots of the probabilities of the SLA actions. (a) The entire simulation. (b)–(e) Zoomed windows on the ranges of interest.

having the left robot pushing alone will be penalized. This is done to avoid having the two robots align parallel to the central line. A similar criterion is used if the object is on the right side of the central line.

In Fig. 18, snapshots of the simulation are shown. Notice that the robots attempt to align the object about the central line in various steps. First, the left robot pushes alone, then both of the robots push at the same time bringing the object closer to the central line. The right robot then pushes the object in order to align it, and then both robots push at the same time bringing the object to the desired location. The probabilities of the actions are demonstrated in Fig. 19. Using SLA, the probabilities

of choosing a certain action is constantly updated based on the feedback from the central station. As shown in Fig. 19, the probabilities converge to the right sequence. Due to the fact that the measurements collected at the base station are corrupted with noise, the actions have to be determined probabilistically.

VI. Conclusion

In most practical problems, the information about the plant to be controlled or the environment in which it operates is not available. In these situations, learning control is needed. SLA provides a means of learning the optimal set of actions in order to achieve the desired goal with minimum penalty. Implementing SLA requires the creation of a simulation platform to model the physics of the problem. The architecture of such a platform is presented as V-Lab.

Using the V-Lab architecture to perform a simulation allows a developer to avoid a lot of the work involved in creating a distributed simulation. Process communication is handled by CORBA; an intelligent methodology for creating objects and their communication is provided by DEVS; and a structure specific for simulations is provided by V-Lab. Furthermore, since V-Lab uses a layered pattern in its design, if any of the layers become obsolete in the future, they 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.

Future work is being considered that would allow high-level models to be transferred from one machine to another dynamically. A simple solution could be to monitor the amount of messages coming into a model from local models and from inter-machine models. If the amount of messages coming in from the external models exceeds the amount of messages coming from internal models, the simulation might benefit from the model being moved to another machine. For distributed simulations using just two machines, the problem of model placement is analogous to graph partitioning, a NP-complete problem [21]. This indicates that as the number of models grows, the amount of time required to find the optimal placement of the models grows at a much larger rate. This, unfortunately, means that no algorithm for a large number of models could be guaranteed to find the optimal solution in every case. However, this is not to say that effective heuristics cannot be designed to determine good solutions to the problem of graph partitioning, as Johnson [22] shows with simulated annealing.

Other future work could involve allowing a simulation to dynamically grow onto more machines. If the SimEnv model found that it was demanding more computational power than it had and was causing other SimEnv models across the network to constantly wait for it to finish, it could spawn a copy of itself on another machine in an attempt to reduce its workload. Logan and Theodoropoulos [23] have come up with multiagent simulations that use this "divide and conquer" method to use multiple processes that determine when they should spawn off children processes to aid in doing work.

In the simulation section of this paper, a case study was presented. The goal was to coordinate two robots in order to push an object to a central location. Without any knowledge about the object to be moved, and with corrupted measurements, the robots are supposed to determine the optimal set of actions. Using SLA, the robots successfully pushed the object to the desired location. Future work will include the use of the hierarchical structure introduced by Saridis [24], [25]. The hierarchical architecture consists of an *organizational level*, a *coordination level*, and an *execution level*. A command is sent to the organizational level (top level); the output of the organizational level is a set of alternative tasks that are capable of executing the command. The coordination level then takes a task as an input and composes set of primitive tasks. Then, the execution level translates the primitive tasks to actions. In order to determine the optimal selection of tasks and primitive tasks, two translation levels are used. This structure will facilitate the design of learning controller that can handle more sophisticated problems. Furthermore, other optimization techniques such as the one proposed by Fathi and Hildebrand [26] will be studied.

Due to the complexity of the project, not all of the V-Lab components were designed, and for the front end of the simulation, Webots software designed by Cyberbotics was used [20]. However, in the future, V-Lab will have its own front end (GUI) of the simulation.

Acknowledgment

The authors thank S. Mallipeddi and Dr. B. Zeigler, University of Arizona, and Dr. H. Sarjoughin, Arizona State University, for their contributions and helpful discussions.

References

[1] J. P. How and M. Tillerson, "Analysis of the impact of sensor noise on formation flying control," in *Proc. Amer. Control Conf.*, vol. 5, 2001, pp. 3986–3991.

[2] P. De Rego, "Increased efficiency in interferometric synthetic aperture radar-InSNAR," Ph.D. dissertation, Dept. Elect. Comput. Eng. and ACE Center, Univ. New Mexico, Albuquerque, Aug. 2002.

[3] H. R. Berenji and D. Vengerov, "Advantages of cooperation between reinforcement learning agents in difficult stochastic problems," in *Proc. 9th IEEE Int. Conf. Fuzzy Systems*, vol. 2, 2000, pp. 871–876.

[4] G. Dudek, M. R. M. Jenkins, E. Milios, and D. Wilkes, "A taxonomy of multiagent robotics," in *Autonomous Robots*. Norwell, MA: Kluwer, 1997.

[5] E. Uchibe, M. Nakamura, and M. Asada, "Coevolution of cooperative behavior acquisition in a multiple mobile robot environment," in *Proc. IEEE Int. Conf. Intelligent Robots and Systems*, vol. 1, 1999, pp. 425–430.

[6] M. Nilli-Ahmadabdadi, M. Asadpur, S. H. Khodaabakhsh, and E. Nakano, "Expertness measuring in cooperative learning," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2000, pp. 2261–2267.

[7] K. S. Fu, "Learning control systems-review and outlook," *IEEE Trans. Automat. Contr.*, vol. AC-15, no. 2, pp. 210–221, 1970.

[8] C. Chandrasekharan and D. W. C. Shen, "On expediency and convergence in variable structure stochastic automata," *IEEE Trans. Syst. Man Cybern.*, vol. SSC-5, pp. 52–60, 1968.

[9] S. Lakshmivarahan, *Learning Algorithms: Theory and Applications*. New York: Springer-Verlag, 1981.

[10] T. J. Gordon, C. Marsh, and Q. H. Wu, "Stochastic optimal control of active vehicle suspensions using learning automata," in *Proc. Mech. Eng.—Part I: J. Syst. Contr. Eng.*, vol. 207, 1993, pp. 143–152.

[11] K. Naruse and Y. Kakazu, "Strategy acquisition of path planning of redundant manipulator using learning automata," in *IEEE Int. Workshop Neuro-Fuzzy Controllers*, 1993, pp. 154–159.

[12] C. Unsal, P. Kachroo, and J. S. Bay, "Multiple stochastic learning automata for vehicle path control in an automated highway system," *IEEE Trans. Syst., Man, Cybern. A*, vol. 29, pp. 120–128, Jan. 1999.
[13] M. Jamshidi and Z. Geng, "A learning 2-D control approach for robot manipulator," *J. Expert Syst. Applicat.*, vol. 4, no. 3, pp. 297–304, 1992.
[14] Z. Geng and M. Jamshidi, "Learning control-system analysis and design based on 2–D system-theory," *J. Intell. Robot Syst.*, vol. 3, no. 1, pp. 17–26, 1990.
[15] Arizona Center for Integrative Modeling and Simulation. (2002) ACIMS Report. [Online]. Available: http://www.acims.arizona.edu/, electronic document.
[16] ——, (2002) DEVS/HLA Tutorial. [Online]. Available: http://www.acims.arizona.edu/DEVS_HLA/devs_hla.html, electronic document.
[17] B. Zeigler. (2002) DEVS theory of quantized systems. [Online]. Available: http://www.acims.arizona.edu/DEVS_HLA/CDRLs/UnivArizonaCDRL1.pdf, electronic document.
[18] The Object Management Group. CORBA BASICS. [Online]. Available: http://www.omg.org/gettingstarted/corbafaq.htm, electronic document.
[19] K. S. Fu and T. J. Li, "Stochastic automata as models of learning systems," in *Computer and Information Sciences*, J. T. Tou, Ed. New York: Academic, 1967, vol. 2.
[20] Cyberbotics. (2002) Webots 3.0 User Manual, Lausanne, Switzerland. [Online]. Available: http://www.cyberbotics.com.
[21] B. Moret and H. Shapiro, *From P to NP*. Reading, MA: Addison-Wesley, 1990, vol. 001.
[22] D. S. Johnson, "Optimization by simulated annealing: An experimental evaluation—Part I: Graph partitioning," *Oper. Res.*, vol. 37, no. 6, pp. 865–892, 1989.
[23] B. Logan and G. Theodoropoulos, "The distributed simulation of multi-agent systems," *Proc. IEEE*, vol. 89, pp. 174–185, Feb. 2001.
[24] K. P. Valavanis and G. N. Saridis, *Intelligent Robotic Systems*. Norwell, MA: Kluwer, 1992.
[25] P. U. Lima and G. N. Saridis, "Intelligent controllers as hierarchical stochastic automata," *IEEE Trans. Syst., Man, Cybern. B*, vol. 29, pp. 151–163, Apr. 1999.
[26] M. Fathi and L. Hildebrand, "Model-free optimization of fuzzy-rule-based system using evolutionary strategies," *IEEE Trans. Syst., Man, Cybern. B*, vol. 27, pp. 270–277, Apr. 1997.

Aly I. El-Osery (S'00–M'02) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of New Mexico, Albuquerque, in 1997, 1998, and 2002, respectively.

From 1997 to 2002, he was a Research Assistant at the Autonomous Control Engineering Center at the University of New Mexico. In 2002, he joined the Department of Electrical Engineering at New Mexico Institute of Mining and Technology, Socorro. His research interests are in the areas of multiagents robotics, wireless communications, control systems, and soft computing.

Dr. El-Osery has received many awards, including Outstanding Junior (1996) and Outstanding Graduate Student (1998) from the Department of Electrical Engineering, and the School of Engineering Award for Outstanding Graduate Student in Electrical and Computer Engineering (1998–1999 and 2001–2002).

John Burge was born in Albuquerque, NM. He received the B.S. degree in computer science from the University of New Mexico, Albuquerque, in 2002, where he is currently pursuing the M.S. degree.

Mo Jamshidi (S'66–M'71–SM'74–F'89) received the B.S.E.E. from Oregon State University, Corvallis, in 1967, and the Ph.D. degree in electrical engineering from the University of Illinois at Urbana-Champaign in 1971, and an honorary doctorate degree from Azerbaijan National University, Baku, in 1999.

Currently, he is the Regents Professor of Electrical and Computer Engineering, the AT&T Professor of Manufacturing Engineering, and Founding Director of the Center for Autonomous Control Engineering (ACE) (http://ace.unm.edu) and co-founded the PURSUE program (http://pursue.unm.edu) at the University of New Mexico, Albuquerque. He was on the advisory board of the NASA Jet Propulsion Laboratory's (JPL) MARS Pathfinder project mission. He is currently a Member of the NASA Minority Businesses Resource Advisory Committee (MBRAC) and was a Member of the NASA JPL Surface Systems Track Review Board. He was on the U.S. National Academy of Sciences NRC's Integrated Manufacturing Review Board and was on NAE's task force on aerospace engineering. Previously, he spent six years at the U.S. Air Force Research (weapons or Phillips) Laboratory working on large-scale systems, control of photonics systems, and adaptive optics. He has been a Consultant with the Department of Energy's Los Alamos National Laboratory, Sandia National Laboratories, and Oak Ridge National Laboratory. He has worked in various academic and industrial positions at many national and international locations including with IBM and GM. As ACE Center Director, he has led a large academic team of researchers and educators which has, thus far, resulted in over 75 M.S. graduates and 23 Ph.D. graduates in engineering. He has over 500 technical publications including 49 books and edited volumes. Six of his books have been translated into at least one foreign language. He is the Founding Editor, Cofounding Editor, or Editor-in-Chief of five journals (including Elsevier's *International Journal of Computers and Electrical Engineering*) and Wiley's and IOS Press's *Intelligent and Fuzzy Systems*, Coeditor-in-Chief since its inception in 1992).

Dr. Jamshidi is a Fellow of the IEEE for contributions to "large-scale systems theory and applications and engineering education," a Fellow of the ASME for contributions to "control of robotic and manufacturing systems," a Fellow of the American Association for the Advancement of Science (AAAS) for contributions to "complex large-scale systems and their applications to controls and optimization," an Associate Fellow of Third World Academy of Sciences (Trieste, Italy), a Member of the Russian Academy of Nonlinear Sciences, an Associate Fellow of the Hungarian Academy of Engineering, an Associate Fellow of Persian Academies of Science and Engineering, and a Member of the New York Academy of Sciences. He was the recipient of the IEEE Centennial Medal, the IEEE Control Systems Society Distinguished Member Award, and the IEEE Control Systems Society Millennium Pin. He is an Honorary Professor at three Chinese universities. He is on the Board of Nobel Laureate Glenn T. Seaborg Hall of Science for Native American Youth. He is the Founding Editor-in-Chief (1980–1984) of IEEE C*ontrol* S*ystems* M*agazine*. He has been on the executive editorial boards of a number of journals and two encyclopedias. He was the series editor for ASME Press Series on Robotics and Manufacturing from 1988 to 1996 and Prentice-Hall Series on Environmental and Intelligent Manufacturing Systems from 1991 to 1998. In 1986, he helped launch a specialized symposium on robotics, which was expanded to International Symposium on Robotics and Manufacturing (ISRAM) in 1988. In 1994, it was expanded into World Automation Congress (WAC) (http://wacong.com) where it now encompasses five main symposia and forums on robotics, manufacturing, automation, control, soft computing, financial engineering, multimedia, and image processing. He has been the General Chairman of WAC from its inception. He is a copublisher of WAC's Official Publication—*Intelligent Automation and Soft Computing* (Albuquerque, NM: TSI Press, 1995–present).

Antony Saba, photograph and biography not available at the time of publication.

Madjid Fathi (S'98–M'92–SM'00) received the B.S. degree from RWTH, Aachen, Germany, and the M.Sc. degree in computer science and the Ph.D. degree in mechanical engineering from the University of Dortmund, Dortmund, Germany, in 1991.

He is currently a Research Professor at the NASA Center for Intelligent System Engineering (ISE) and the Center of Autonomous Control Engineering (ACE) at the University of New Mexico, Albuquerque. He, together with Dr. Temme, have invented the Fix-Mundies Theory for minimizing side-effects of hypertension patients. The technology is now being prepared for a major pharmaceutical corporation to produce a number of cocktail drug products. The technology may also be applied to mechanical systems by enhancing stability processes. The work will benefit other applications with on-line configuration tasks.

Mohammad-R. Akbarzadeh-T. (M'98–SM'02) received the B.S., M.S., and Ph.D. degrees in electrical engineering, with concentration in robotics and control system, all from the University of New Mexico, Albuquerque, in 1989, 1992, and 1998, respectively. His dissertation addressed the utility of fuzzy logic and genetic algorithms in control of complex systems.

From 1996 to 1999, he held dual appointments as Research Engineer at the Center for Autonomous Control Engineering, as well as Adjunct Assistant Professor in the Department of Electrical Engineering at the University of New Mexico, where he was responsible for various research involving applications of soft computing in systems such as flexible robots, multistage flash desalination process, and multiagent robotic systems. Since 1999, he has been with the Faculty of Electrical Engineering at Ferdowsi University of Mashhad, Mashhad, Iran. His current research interests include intelligent systems, nano-robotics, multiagent robotic learning/cooperation, and soft computing.