Implementation of a Distributed Problem Solving Framework based on Discrete Event System Specification

Heung Bum Kim, Yeong Rak Seong, Tag Gon Kim and Kyu Ho Park

Computer Engineering Research Laboratory
Department of Electrical Engineering
Korea Advanced Institutes of Science and Technology
373-1 Kusong-dong Yusong-gu, Taejon 305-701,
Korea.

Abstract

Distributed Problem Solving(DPS) is a field of distributed artificial intelligence. In a DPS system, a problem is solved by a team of cooperating intelligent agents, each intelligent agent having partial data and knowledge of the problem and the problem solving environment. A DPS system is a discrete event system. In the problem-solving process, an intelligent agent randomly receives and processes messages from the outside and other intelligent agent; it also unpredictably activates knowledge sources in itself.

This paper proposes a DPS framework based on DEVS formalism specifying discrete event systems. The framework consists of two components, AGENT and WHITEBOARD. The AGENT maintains several problem-solving plans. It is able to solve many problems concurrently. The WHITEBOARD keeps *capability* and *loading factor* of AGENTs and distributes messages during the solving process. For verifying the proposed framework, an example system with three intelligent agents was examined. The results show that the DPS framework concurrently solves problems, and each AGENT is maintained appropriate load.

1 Introduction

Distributed Problem Solving(DPS) is an area of distributed artificial intelligence. It examines how a loosely coupled network of intelligent agents(problem-solving nodes) can solve problems that are beyond the capabilities of the agents individually[1]. Furthermore, although an intelligent agent can solely solve the problems, such distributed/parallel processing is still useful for reducing the overall finishing time. Each agent is a sophisticated system that can modify its behavior as circumstances change and plan its own communication and cooperation strategies with other intelligent agents. Although DPS borrows ideas from numerous fields, including distributed processing, artificial intelligence, and the social sciences, it differs significantly from each of those in the problems being attacked and the methods used to solve them.

Many DPS systems have appeared over the past several years in artificial intelligence and simulation fields. They become more mature systems in various disciplines. Hearsay-II is a continuous speech understanding system developed at Carnegie-Mellon University[2]. Its architecture is better known than the speech understanding system itself. McArthur *et al.* [3] has been studying the problem of distributed air traffic control for several years. This is a distributed planning problem consisting of a simulated airspace with planes. A distributed vehicle monitoring testbed builds a simulated environment with moving vehicles that generate sounds which can be picked up by acoustic sensors[4]. The goal of the distributed problem solver is to create a dynamic map of the area covered by the sensors. The Contract Net proposed by Davis and Smith is the most common message-based functionally accurate negotiation-style protocol. An interesting practical example is the use of contract nets in a flexible manufacturing system by Parunak[5]. In simulation area, Nadoli[6] implements an intelligent manufacturing-simulation agent tool. Most DPS systems proposed in the previous researches cannot solve more than one problem concurrently. They are not specified in modular, hierarchical manner.

The DPS system is a discrete event system. The intelligent agent interacts with the outside through randomly generated or arrived messages. It processes arrived messages with respect to its local information. Thus, knowledge sources are unpredictably activated.

The DEVS(Discrete Event Systems Specification) formalism, developed by Zeigler[7], provides a methodology for specifying complex discrete event systems in a hierarchical, modular manner. It also provides a basis for sound semantics for developing discrete event simulation languages and environments. DEVSim++ is an example of such environment. It realizes the formalism in the object-oriented environment of C++ [8].

In this paper, we propose a DPS framework based on the DEVS formalism. Within the proposed framework, many problems can be solved concurrently. Thus, an intelligent agent maintains several problem-solving plans. Since the DEVS formalism separates the structure from the behavior of the modeled system, the proposed framework is classified by structure and behavior. Also, it is specified in a modular, hierarchical manner. Currently, it is implemented and operated in the DEVSim++ environment.

0-7803-2559-1/95 $4.00 © 1995 IEEE

The paper is organized as follows. Section 2 briefly introduces the DEVS formalism. In Section 3, our DPS framework is specified according to the DEVS specification. For verifying the proposed framework, an example system with three intelligent agents was examined in Section 4. In this experiment, the problem is represented by AND/OR tree. Section 5 states our conclusion.

2 DEVS Formalism

A discrete event system can be specified by Zeigler's DEVS formalism[7]. Within the formalism, one specifies a discrete event system in two kinds of models: atomic model and coupled model. An atomic model is an unbreakable component of a system, which is represented as follows.

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, t_a >$$

where

X : input events set, finite set;

Y : output events set, finite set;

S : sequential states set, finite set;

δ_{ext} : Q x $X \rightarrow S$
: external transition function;

δ_{int} : $S \rightarrow S$: internal transition function;

λ : $S \rightarrow Y$: output function;

t_a : $S \rightarrow$ Real : time advanced function.

where $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$: total state of M.

The first three elements, in the 7-tuple are the system's input, state set, and output, and the next four elements give the constraints among the three.

A coupled model is a compound component consisting of atomic models and/or coupled models. The coupled model can itself be employed as a component in a larger coupled model, thereby giving rise to the construction of complex models with hierarchical structures. A coupled model is represented as follows.

$$DN = < X, Y, M, EIC, EOC, IC, SELECT >$$

where

X : input events set, finite set;

Y : output events set, finite set;

M : set of all component models, finite set;

EIC : $EIC \subseteq DN.IN$ x $M.IN$
: external input coupling relation;

EOC : $EOC \subseteq M.OUT$ x $DN.OUT$
: external output coupling relation;

IC : $IC \subseteq M.OUT$ x $M.IN$
: internal coupling relation;

$SELECT$: $2^M - 0 \rightarrow M$
: tie-breaking selector.

DEVSim++ is a realization of the DEVS formalism and associated object-oriented hierarchical simulation algorithms in C++ [8]. This environment supports modelers in developing discrete event models using the hierarchical composition technology within the DEVS framework.

3 DPS Framework

Problem solving in a DPS system is divided into three phases. In the first phase, a larger problem is decomposed into a set of smaller subproblems. This step is repeated until all the subproblems can no longer be decomposed. In the second phase, each subproblem is solved. During this phase, intelligent agents communicate and cooperate with each other. Answers are synthesized in the third phase. Thus, sub-results are integrated to obtain a solution to the overall problem. The aim of DPS is to resolve all the phases effectively.

Three issues are involved in the construction of a DPS system [9]: 1) problem distribution, 2) control mechanism, and 3) communication policy. To solve problems effectively, each subproblem is evenly assigned to each intelligent agent. Otherwise, some extra useless communication can be inserted. The control mechanism maintains a global coherence during the problem solving process. Moreover, computing loads should be distributed as even as possible for higher overall performance. The communication policy defines interaction between intelligent agents. Intelligent agent must cooperate with others when none of them has sufficient information to solve the entire problem. Such information sharing allows the system to produce a consistent answer.

Our DPS framework is designed under two important assumptions. First assumption is that the communication cost among cooperating intelligent agents is much smaller than the computing cost to solve each problem. Occasionally, a DPS system can be classified from general distributed systems by its abstraction level of data communicated. In a DPS system, the data communicated are highly abstracted. Thus, the total amount of data communicated is not very large. Second assumption is that problems are continuously entered into the DPS system from the outside. Hence, several problems would be processed concurrently in the proposed DPS system.

According to the above first assumption, a centralized communication scheme is employed in our DPS framework. In the scheme, all intelligent agents are connected to a whiteboard, a centralized global coordinator. The whiteboard is a counterpart of a blackboard. Generally, in a DPS system, a blackboard is a passive data structure. It stores tentative results and various environmental informations. Data are referred to by intelligent agents. A whiteboard corresponds to an active blackboard. In addition to the features of a blackboard, it has the function of administering intelligent agents. Hence, it forwards incoming messages to appropriate intelligent agents. To determine such intelligent agents, a whiteboard maintains two kinds of information about each intelligent agent that is connected to whiteboard: *capability* and *loading factor*.

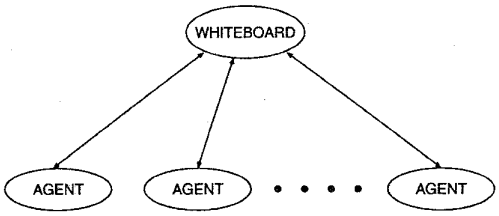

Figure 1: A Framework for DPS

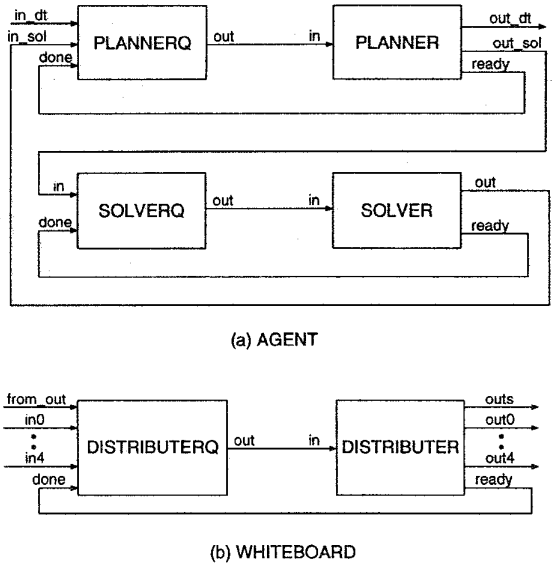

Figure 2: Coupled models of DPS Framework

Capability specifies the types of knowledge(the list of solvable problems) which is determined at the beginning of bootstrapping time; loading factor represents the current computing load which is continuously set on the receiving messages from the intelligent agent. By using the information, a whiteboard selects an intelligent agent that has the proper knowledge with minimal computing load.

As mentioned earlier, the proposed framework is composed of two kinds of components: WHITEBOARD and AGENT. WHITEBOARD is a whiteboard which stores tentative results and various kinds of environmental information, and administers AGENTS. AGENT is an intelligent agent which stores knowledge for solving problems. Figure 1 shows the interconnection between the components in the proposed DPS framework. As mentioned earlier, the framework employs a centralized architecture. Thus, all AGENTs are connected to one WHITEBOARD. The two components are described as coupled models of the DEVS formalism. There are two types of components; WHITEBOARD and AGENT. Their internal structures are shown in Figure 2. AGENT consists of four components: PLANNERQ, PLANNER, and SOLVERQ

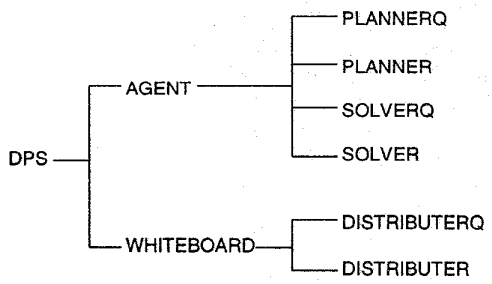

Figure 3: Components of DPS Framework

and SOLVER. WHITEBOARD is composed of two components: DISTRIBUTERQ and DISTRIBUTER. Figure 3 depicts the hierarchical construction of the entire framework. In the tree, a leaf nodes is specified by atomic DEVS. An internal nodes is specified by coupled DEVS.

The components share no variables and communicate with others through message passing only. The message has four types: question, answer, load, and initial. A question type message holds the problem to be solved; an answer type message contains the solution of a problem previously requested. A load type message specifies the current loading factor of an AGENT; and an initial type message describes the capability of an AGENT.

The atomic models except PLANNER model are briefly explained as follows:

- PLANNERQ: receive message from WHITEBOARD or SOLVER, sort by priority, and if PLANNER is free, send message to it.
- SOLVERQ: receive message from PLANNER, sort by priority, and if SOLVER is free, send message to it.
- DISTRIBUTERQ: receive message from other AGENTs and the outside or DISTRIBUTER, sort by priority, and if DISTRIBUTER is free, send message to it.
- SOLVER: solve self-solvable problem by using knowledge source.
- DISTRIBUTER: provide messages to appropriate AGENT by using the information of capability and load condition.

PLANNER is invoked when a message needs to be planned. PLANNER has its own local meta-knowledge. It searches the meta-knowledge for a suitable problem-decomposition and then generates a problem-solving plan.

Figure 4 shows the DEVS modeling result of PLANNER. PLANNER has one input port("in") and three output ports("out_dt", "out_sol", "ready"). Each input port is represented by an incoming arrow and each output port is represented by an outgoing arrow. "in" is connected to PLANNERQ. "out_dt", "out_sol" and "ready" are linked to WHITEBOARD, SOLVERQ, and PLANNERQ, respectively. Also, PLANNER has seven state variables:

"phase", "buf", "plan", "knowledge", "dt", "sol", and "name". Each state variable is shown in the small box in Figure 4. The "phase" describes the current execution mode of PLANNER. As shown in the phase transition and output generation diagram in the outer box, PLANNER has four phases: BEGIN, WAIT, PROCESS, and END. Details of the phases will be explained later. "buf" is an input buffer for saving incoming messages via "in". "dt" and "sol" are the output buffers for saving outgoing messages via "out_dt" and "out_sol", respectively. "plan" maintains the problem-solving plans in the active problem solving process. "knowledge" is meta-knowledge. Also, "name" represents the name of PLANNER.

Now, let's discuss the phase transition and output generation of PLANNER in Figure 4. In the figure, the arrival of an input message is specified by "?"; generation of an output message is represented by "!" and a dotted line. For example, "?answer" means that a message with the name of "answer" arrives at an input port of PLANNER; "!out_dt" indicates that a message with the name of "out_dt" is sent to the outside through an output port of PLANNER. In the DEVS formalism, there are two kinds of phase transition. External transition is specified by solid arrows; internal transition is elaborated by dashed arrows. For example, the solid arrow from WAIT to PROCESS with the label "?answer" specifies the arriving of "answer" message in WAIT phase. Phase transition may depend on state variables, except "phase". In such cases, "@" character is used. For example, the dashed line from END to WAIT with the label "@buf=empty" means that if PLANNER executes internal transition while "buf" is empty, phase is changed from END to WAIT.

Initially, PLANNER is in BEGIN phase. As soon as the execution is started, PLANNER sends a message describing its capability toward WHITEBOARD through the "out_dt" output port and changes its phase to WAIT. When PLANNER receives an external event in WAIT, it changes its phase to PROCESS. In "PROCESS" phase, PLANNER has two operational modes: initial planning mode and replanning mode. The operational mode is determined by the type of the received external event message. Thus, if the type is question, the mode is initial planning mode. Otherwise, the mode is replanning mode. In the initial planning mode, this model generates a problem-solving plan by using the information of meta-knowledge. Based on the problem-solving plan, messages are created. All unsolvable kernel subproblems are transmitted to WHITEBOARD while self-solvable kernel subproblems are sent to SOLVERQ. In the replanning mode, PLANNER deduces the problem-solving plan. If the contents of a message are the results of a kernel subproblem, PLANNER updates the meta-knowledge "knowledge". If the problem-solving plan is deducible according to the received message, it is modified repeatedly. When a problem is solved, PLANNER generates a message containing the result of the problem, and transmits into WHITEBOARD. Especially, if PLANNER has no plan to process, it sends a "load-type" message describing the load factor to WHITEBOARD for balancing load and receiving new initial problem.

4 Experiments and Results

The proposed DPS system is implemented within the DEVSim++ environment. To verify the proposed DPS framework, an example DPS system with one WHITEBOARD and three AGENTs(AGENT1, AGENT2, and AGENT3) is organized. All AGENTs are connected to WHITEBOARD and communicates with it. WHITEBOARD is also in charge of communicating with the outside.

Figure 5 shows that a problem-decomposition tree used in this experiment is an AND/OR tree. AND node is deduced when all its subproblems of AND node are solved. OR node is deduced when one subproblem of OR node is solved. In an AND/OR tree, each leaf node is a kernel subproblem, and each internal node is an internal problem. In Figure 5, the tree consists of eight kernel subproblems and six internal problems. For simplicity, we assume that each AGENT has the same capability. In this case, an AGENT can solve a problem without any cooperation with other AGNETs. In order to reduce the overall finishing time, problems are cooperatively solved by the team of AGENTs. However, the cost may vary with respect to which AGENT solves the problem. The numbers attached to each leaf node show the costs. For example, the processing cost of A2 is 1, 2, and 3 when it is processed by AGENT1, AGENT2, and AGENT3, respectively. By the semantics of an AND/OR tree, the overall processing cost of an AND node is the sum of its subordinates' cost; that of an OR node is the minimum of its subordinates' cost. The number enclosed by parentheses represents the optimal cost of the associated problem.

Problems are the inputs from the external environment to the DPS framework. In this experiment, Prob1, Prob2, Prob3 and Prob4 are continuously generated and entered into the experimental system at t = 10, 13, 512, and 515. The problem types of Prob1 and Prob3 are A1; that of Prob2 and Prob4 are A7.

The simulation is executed under the DEVSim++ environment. The proposed system solves the given problems well. Table 1 shows the problem solving history of Prob2. When Prob2 is generated, Prob1 is being solved. Thus, the two problems are solved concurrently. However, Table 1 does not show the problem solving history of Prob1. After receiving Prob2 from WHITEBOARD, AGENT2 makes the problem-solving plan for the problem. First, since Prob2 is A7 type, it is decomposed into A9 and A10, which are represented by Prob2.A9 and Prob2.A10, respectively. The two subproblems are also decomposed. Consequently, Prob2 is decomposed into Prob2.A9.A11, Prob2.A9.A12, Prob2.A10.A13, and Prob2.A10.A14. For the subproblems, four messages are generated and transferred to WHITEBOARD. At that time, because the loading factor of AGENT2 is higher than those of AGENT1 and AGENT3, WHITEBOARD decides to forward the messages toward AGENT1 and AGENT3. After AGENT1

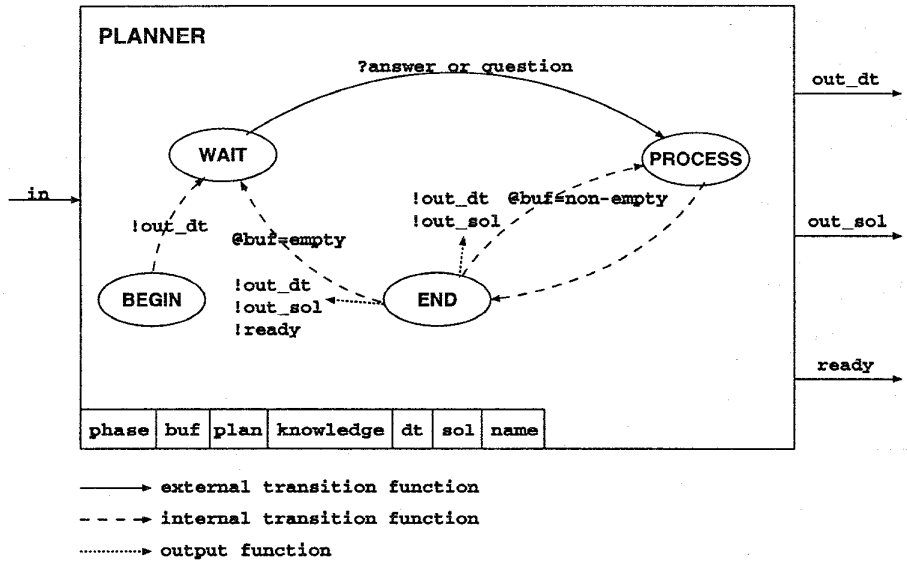

Figure 4: An Atomic Model of PLANNER

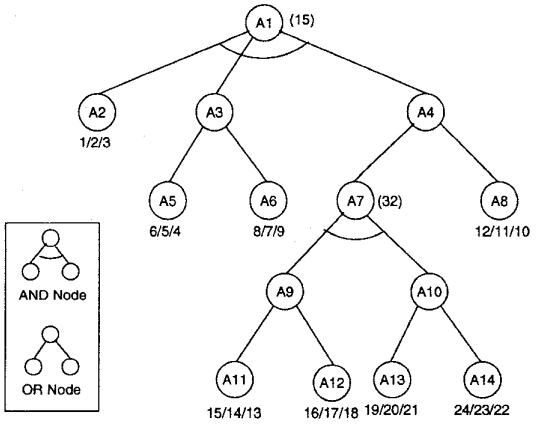

Figure 5: Problem Decomposition Tree for Experiment

and AGENT3 solve the problems, the results are returned back to AGENT2. Finally, AGENT2 integrates the results and finds out the overall solution.

Table 2 shows the statics of the overall experiment. In the table, "Solved IA" is the intelligent agent which deduces the overall solution; "Task" and "Message" represent the number of messages generated for solving the experiment. The result shows that the solved order differs from the generated order of problems.

5 Conclusion

In this paper, a DPS framework based on the DEVS formalism is proposed. Also, the proposed framework is

Table 2: Results of Three AGENTs

Solved Sequence		Prob1	Prob2	Prob3	Prob4
Solved AGENT		AG2	AG1	AG2	AG1
Cost		34	15	32	37
Start Time		13	10	515	512
Solved Time		193	493	386	450
AGENT1	Task	0	21	4	6
	Message	0	9	4	2
AGENT2	Task	9	5	7	0
	Message	5	5	7	0
AGENT3	Task	3	7	2	0
	Message	3	7	2	0

implemented and executed in the DEVSim++ environment. The framework is designed with two important assumptions. First assumption is that the communication cost among cooperating intelligent agents is much smaller than the computing cost required for solving each problem. Second is that several problems would be processed concurrently. According to the assumptions, a centralized communication scheme is employed in which all intelligent agents are connected to a whiteboard, a centralized global coordinator. A whiteboard stores tentative results and various environmental informations, forwards messages to intelligent agents. To determine the most appropriate intelligent agent for a problem, a whiteboard maintains capability and loading factor of each intelligent agent which is connected to the whiteboard. Capability specifies the types of knowledge(the list of solvable problems) which is determined at the beginning of bootstrapping time; loading factor represents the current computing load which is continuously set on receiving messages from the intelligent

Table 1: Solving Sequence of Prob2 Problem in First Trial: ($NAME^{type}_{(cost)}$)

WHITEBOARD	AGENT1	AGENT2	AGENT3
$Prob2^{Q}$			
		$Prob2^{Q}$;in	
$Prob2.A10.A13^{Q}$ $Prob2.A10.A14^{Q}$ $Prob2.A9.A11^{Q}$ $Prob2.A9.A12^{Q}$			
	$Prob2.A10.A13^{Q}$		
			$Prob2.A10.A13^{Q}$
	$Prob2.A10.A14^{Q}$		
			$Prob2.A10.A14^{Q}$
	$Prob2.A9.A11^{Q}$		
			$Prob2.A9.A11^{Q}$
	$Prob2.A9.A12^{Q}$		
			$Prob2.A9.A12^{Q}$
		$Prob2.A10.A13^{A}_{(20)}$;so	
		$Prob2.A10.A13^{A}_{(20)}$;re	
		$Prob2.A10.A14^{A}_{(23)}$;so	
		$Prob2.A10^{A}_{(20)}$;re	
		$Prob2.A9.A11^{A}_{(14)}$;so	
		$Prob2.A10.A14^{A}_{(23)}$;up	
		$Prob2.A9.A12^{A}_{(17)}$;so	
		$Prob2.A9.A11^{A}_{(14)}$;re	
		$Prob2.A9^{A}_{(14)}$;re	
		$Prob2^{A}_{(34)}$;re	

agent.

For verifying the proposed framework, an example system with three intelligent agents was examined. In this experiment, the problem is represented by AND/OR tree. The results show that the DPS framework works well; it is able to solve several problems concurrently. Moreover, each AGENT is maintained with appropriate workload.

References

[1] Victor R. Lesser, "A retrospective view of FA/C distributed problem solving", *IEEE Transactions on System, Man, and Cybernetics*, vol. 21, no. 6, pp. 1347–1362, Nov./Dec. 1991.

[2] L. D. Erman, F. Hayes-Roth, V. R. Lesser and D. R. Reddy, "The hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty", *Computing survey*, vol. 12, no. 2, pp. 213–253, June 1980.

[3] D. McArthur, R. Steeb and S. Cammarata, "A framework for distributed problem solving", in *Proc. 2nd Nat. Conf. Artificial Intelligence*, 1982, pp. 181–184.

[4] E. H. Durfee, V. R. Lesser and D. D. Corkill, "Coherent coperation among communication problem solvers", *IEEE Transactions on Computers*, vol. C-36, no. 11, pp. 1275–1291, Nov. 1987.

[5] H. Van Parunak, "Manufacturing experience with the contract net", in *Proc. Distributed Artificial Intelligence Workshop*, 1985, pp. 67–91.

[6] G. Nadoli and J. E. Biegel, "Intelligent manufacturing-simulation agents tool(imsat)", *ACM transaction on Modeling and Computer Simulation*, vol. 3, no. 1, pp. 42–65, Jan. 1993.

[7] Bernard P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, Orlando, FL, 1984.

[8] Tag G. Kim, *DEVSim++ User's Manual: C++ Based Simulation with Hierarchical Modular DEVS Models*, Computer Engineering Lab., Dept. of Electrical Engineering, KAIST, 1994.

[9] Alan H. Bond and Les Gasser, *Reading in distributed artificial intelligence*, Morgan Kaufmann Pub., San Mateo, CA, 1988.