Proceedings of 2000 Conference on AI, Simulation and Planning in High Autonomy Systems, pp.53-59, Mar., 2000, U.S.A.

FRAMEWORK FOR MODELING/SIMULATION OF MOBILE AGENT SYSTEMS

Jae-Hyun Kim and Tag Gon Kim

Department of Electrical Engineering & Computer Science
KAIST
Taejon, KOREA
E-mail: {jhkim,tkim}@smslab.kaist.ac.kr

KEYWORDS

Mobile Agent, Mobile Discrete Event System Specification, Agent Modeling/Simulation.

ABSTRACT

The paper presents the formal modeling and simulation framework for mobile agent systems. A Mobile Discrete Event System Specification (MDEVS) formalism is proposed to represent the dynamics of mobile agent systems. The MDEVS formalism supports the structural changes of systems that include creation, addition, deletion, and migration of models and the dynamic changes of couplings between models. AgentSim is developed, which is a simulation environment for mobile agent systems modeled by the MDEVS formalism. AgentSim is implemented using the contemporary mobile agent development toolkit – IBM's Aglets Software Development Toolkits. An example demonstrates modeling and simulation of an e-commerce mobile agent system using the AgentSim environment.

1 INTRODUCTION

From the remote procedure calling (RPC) to the mobile objects, distributed and network computing paradigms have rapidly evolved for the last few decades. Encouraged by the evolution of technology including the appearance of the platform-independent language, Java, the paradigm finally reached to a mobile agent concept.

This technology is in the first generation, lacking maturity and standards. The rapid evolution has been poorly supported with formal methods. This lack of formal methods causes significant problems especially when researchers need to simulate the mobile agent systems to verify their ideas. Most papers in this field, present their ideas and the results of simulation without the procedures of validation and verification of the models due to lack of formal methods. A framework of modeling and simulation for the mobile agent systems helps every reader to verify the presented models by himself if they are described with the formalism, and even to simulate them again. The framework will also encourage collaboration among researchers.

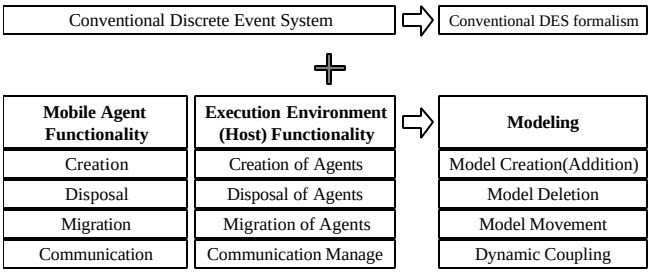

Figure 1: Requirements for Mobile Agents Modeling

Figure 1 shows the requirements for modeling the mobile agent systems. The basic functionality of mobile agents is creation, disposal of agents, migration to other hosts, and communication with other agents. The execution environments of mobile agents actually perform the above functions when receiving requests from agents. Thus a formalism is necessary to support dynamic model creation, deletion, movement, and coupling in order to specify the mobile agent systems.

Recently, there are movements to model and simulate the mobile agents and/or objects (Barros 1999; Uhrmacher 1999). These work propose the formalism or the simulation environment for mobile agent systems. We propose the framework of modeling and simulation for mobile agent systems. Moreover, we focus reducing the gap between physical mobile agent systems and the models of the mobile agents.

The paper is organized as follows. Section 2 presents the formalism for modeling of the mobile agents systems. Section 3 describes the abstract simulation algorithm for the

proposed formalism. Section 4 shows the AgentSim simulation environment for mobile agent systems. Section 5 presents an example e-commerce system. Finally, section 6 concludes the paper.

2 MOBILE DISCRETE EVENT SYSTEMS SPECIFICATION`

This section proposes Mobile Discrete Event System Specification (MDEVS) formalism for modeling, analysis, and simulation of mobile agent systems, which is an extension of the DEVS formalism (Zeigler 1984).

The MDEVS formalism provides the formal method of modeling mobile agent systems by supporting the structural changes of the systems, which include the creation, addition, deletion, and migration of models and the dynamic changes of couplings between models.

2.1 Atomic Model

A basic model, called an atomic model, has specifications for the dynamics of the model. It describes the behavior of a component, which is indivisible, in a timed state transition level. An atomic model of the proposed MDEVS formalism is the same as original one. Formally, the following 7-tuple (Zeigler 1984) specifies an atomic model AM:

$$\mathbf{AM = < X, Y, S, d_{ext}, d_{int}, l, ta >}$$

with the following constraints:
X : Input Events Set;
Y : Output Events Set;
S : States Set;
δ_{ext} : $Q \times X \to S$, External Transition Function;
where $Q = \{(s,e) \mid s \in S, 0 \le e \le ta(s)\}$: Total state of the model;
δ_{int} : $Q \to S$, Internal Transition Function;
λ : $Q \to Y$, Output Function;
ta : $S \to R^{+}_{0,\infty}$, Time Advance Function.

2.2 Coupled Model

The second form of the model, called a coupled model, tells how to couple several component models to form a new model. The coupled model can be employed as a component in a larger coupled model, thus giving rise to the construction of complex models in a hierarchical fashion.

A coupled model of the MDEVS formalism is capable of changing the inside structure. Formally, the following 8-tuple specifies a coupled model CM:

$$\mathbf{CM = <X, Y, S, r, d, \{M_i\}, C, SELECT>}$$

with the following constraints:
$X = X_{in} \cup X_{ch}$,
X_{in} : Input Events Set,
X_{ch} : Structure-Change Events Set;
Y : Output Events Set;
S : Structure-States Set;
$\{M_i\} \in M^{*}$: Activated Models Set,
where M^{*} : Total Models Set;
ρ : $S \to 2^{M^{*}}$, Model Activation Function;
δ : $X_{ch} \times S \to S$, Structure Transition Function;
C = { EIC, EOC, IC, SCC }: Couplings Set;
$EIC \subseteq X_{in} \times \cup_i X_i \times S$, External Input Coupling Relation;
$EOC \subseteq \cup_i Y_i \times Y \times S$, External Output Coupling Relation;
$IC \subseteq \cup_i Y_i \times \cup_j X_j \times S$, Internal Coupling Relation;
$SCC \subseteq \cup_i Y_i \times X_{ch} \times S$, Structure-Change Coupling Relation;
SELECT : $2^{\{M_i\}} - f \to M_i$, Select Function.

The coupled model changes its structure by receiving an input event specified in structure-change events set X_{ch}. The structure-state of the coupled model represents the current structure of the model. The structure-state of the model determines the activated models set and the activated couplings. A received structure-change event changes the structure-state of the model according to the structure transition function. All coupling relations specified above are activated according to the structure-state of the model.

To build complex models formalisms are necessary to support hierarchical and modular models. Models can thus be decomposed into other models in a recursive way. This kind of modular construction is possible only if formalisms are *closed under coupling*.

Definition 1 (Zeigler 1984) A formalism is said to be *closed under coupling* if any composite systems obtained by coupling components specified by the formalism is itself specified by the formalism.

Theorem 1 The MDEVS formalism is closed under coupling; that is, a coupled model CM is mapped into an atomic model AM.

Proof The association mapping CM into AM demonstrates the closure of the formalism under coupling. We will describes AM = < X_A, Y_A, S_A, δ_{ext}, δ_{int}, λ, ta > in terms of the elements in the CM = <X_C, Y_C, S_C, ρ, δ, $\{M_i\}$, C, SELECT>.

First, the input/output events set and the states set of AM are defined as follows; $X_A = X_C$, $Y_A = Y_C$, $S_A = (\times_i S_i) \times S_C$ where S_i is a state set of $M_i \in \{M_i\} = r(s_C)$, and $s_C \in S_C$.

We define s_i by $s_i = ta_i(s) - e_i$ where ta_i is a time advance function of M_i, $s_i \in S_i$ is the current state of M_i, and e_i is a elapsed time of M_i.

Then, the time advance function of AM is defined by $ta(s) = min\{s_i \ \ for\ all\ M_i \in \{M_i\} = r(s_C)\}$.

To define the transition functions, let the current state $s \in S_A$ becomes the new state $s' \in S_A$ by transition.

By the external transition function $s' = d_{ext}(s,e,x)$, each component of the state *s* is transited to the new component as follows:

$$s' = (s'_C, s'_i),\ s = (s_C, s_i)$$

$$s'_C = \begin{cases} d(s_C, x) & if\ x \in X_{ch} \\ s_C & otherwise \end{cases}$$

$$s'_i = \begin{cases} d_{ext,i}(s_i, e_i, x_i) & if\ (x, x_i, s_C) \in EIC, \\ & M_i \in r(s_C) \cap r(s'_C) \\ s_i & if\ (x, x_i, s_C) \notin EIC, \\ & M_i \in r(s_C) \cap r(s'_C) \\ s_{i,0} & if\ M_i \in r(s'_C) - r(s_C) \end{cases}$$

where $s_C, s'_C \in S_C$, s_i is a state of M_i and s'_i is a state of M'_i. Note that $r(s_C) = \{M_i\}$ and $r(s'_C) = \{M'_i\}$.

By the internal transition function $s' = d_{int}(s)$, each component of the state *s* is transited to the new component as follows:

$$s'_C = \begin{cases} d(s_C, x) & if\ (y_i, x, s_C) \in SCC, y_i = l_i(s_i), s_i = 0 \\ s_C & otherwise \end{cases}$$

$$s'_i = \begin{cases} d_{ext,i}(s_i, e_i, x_i) & if\ (y_j, x_i, s_C) \in IC, y_j = l_j(s_j), \\ & s_j = 0, M_i \in r(s_C) \cap r(s'_C) \\ d_{int,i}(s_i) & if\ s_i = 0, M_i \in r(s_C) \cap r(s'_C) \\ s_{i,0} & if\ M_i \in r(s'_C) - r(s_C) \\ s_i & otherwise \end{cases}$$

The output function is defined as

$y = l(s) = l_i(s_i)\ if\ (y_i, y, s_C) \in EOC, M_i \in r(s_C)$

Q.E.D

2.3 Example

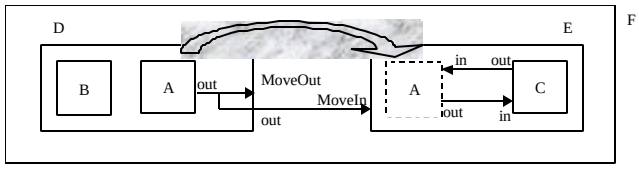

Figure 2: Migration of **A** from **D** to **E**

This section describes a modeling example with MDEVS formalism. Figure 2 shows the example system. Model **A** is moving from model **D** to model **E** by sending an event through **out** port. **E** deletes **A** when it receives the **MoveOut** event, and **D** adds **A** when it receives the **MoveIn** event. Note that the state of **A** should be invariant during a migration process.

Model **D** and **E** are specified as follows:

$D = < X, Y, S, \{M_i\}, r, d, C, SELECT >$

$X = X_{ch} = \{MoveOut\};$

$Y = \{out\};$

$S = \{A\_IN, A\_OUT\};$

$?(A\_IN) = \{A,B\},\ ?(A\_OUT) = \{B\};$

$d(MoveOut, A\_IN) = A\_OUT;$

$EOC = \{(A.out, out, A\_IN)\};$

$SCC = \{(A.out, MoveOut, A\_IN)\};$

$SELECT(\{A,B\}) = A.$

$E = < X, Y, S, \{M_i\}, r, d, C, SELECT >$

$X = X_{ch} = \{MoveIn\};$

$S = \{A\_IN, A\_OUT\};$

$?(A\_IN) = \{A,C\},\ ?(A\_OUT) = \{C\};$

$d(MoveIn, A\_OUT) = A\_IN;$

$IC = \{(A.out, C.in, A\_IN), (C.out, A.in, A\_IN)\};$

$SELECT(\{A,C\}) = A.$

3 ABSTRACT SIMULATORS ALGORITHM

The MDEVS formalism uses the abstract simulator concept for simulation. Zeigler proposed the abstract simulator concept (Zeigler 1984) for simulation of DEVS models. This concept associates each model to a virtual (or simulation) processor that interprets the dynamics specified by the formalism, in a one-to-one manner. A simulation proceeds by means of message passing among abstract simulators, not among DEVS models.

Table 1: Messages for Abstract Simulators

Type	Name	Send	Receive	Meaning
(*,t)	Star	RC,C	C,S	Internal event(*) at time t
(x,t)	X	C	C,S	External event(x) at time t
(y,t)	Y	C,S	C	Output(y) at time t
(done,t_N)	Done	C,S	RC,C	Event handling is done, and next scheduling time is t_N
(query)	Query	RC,C	C,S	Query the next scheduling time

S : Simulator, C : Coordinator, RC : Root Coordinator

There are two types of processors: a *simulator* for an atomic model and a *coordinator* for a coupled model. A special kind of the coordinator called *root coordinator* is associated with no models and takes responsibility for ad-

vancing simulation time. Each processor simulates a system by sending and/or receiving the five types of messages in Table 1. The first four types of messages originated from the abstract simulation algorithm of DEVS formalism. The *query* message is added in order to update next scheduling time of newly added or immigrated models.

3.1 Simulator for Atomic Models

The job of an abstract simulator for an atomic model is to schedule the next event time and to request the associated atomic model to execute its transition functions and output function timely. The event handling algorithms for the abstract simulator are as follows:

```
Simulator:when_rcv_(x,t)
t_L ≤ t ≤ t_N
1. e := t – t_L;
2. s := δ_ext(s,e,x);
3. t_L := t;
4. t_N := t_L + ta(s);
5. send (done,t_N) to the parent coordinator;

Simulator:when_rcv_(*,t)
t = t_N
1. y := λ(s);
2. send (y,t) to the parent coordinator;
3. s := δ_int(s);
4. t_L := t;
5. t_N := t_L + ta(s);
6. send (done,t_N) to the parent coordinator;

Simulator:when_rcv_(query)
1. t_N := t_L + ta(s);
2. send (done,t_N) to the parent coordinator;
```

3.2 Coordinator for Coupled Models

The responsibilities of the abstract simulator for a coupled model is to synchronize the component abstract simulators for scheduling the next event time and to route external event messages to component simulators. Scheduling and event routing are performed in a hierarchical manner. A coordinator for the coupled model also takes responsibility for the structural modification. The event handling algorithms for the coordinator are as follows:

```
Coordinator:when_rcv_(x,t)
t_L ≤ t ≤ t_N
1. if x∈X_in
2.   for each affected processor p^i,
3.     send (x,t) to p^i;
4.     wait for (done,t_N)^i from p^i;
5.   end
6. else
7.   s := δ(x,s);
8.   M_prev := {M_i};
9.   {M_i} := ρ(s);
10.  for all processors p^j in {M_i}-M_prev
11.    send (query) to p^j;
12.    wait for (done,t_N)^j from p^j;
13.  end
14.end
15.t_L := t;
16.t_N := min { t_N of all child processors };
17.send (done,t_N) to the parent coordinator;

Coordinator:when_rcv_(*,t)
t = t_N
1. find the imminent processors with min t_N;
2. select one, p*, and send (*,t) to it;
3. wait for (done,t_N)* from p*;
4. for all received (y,t)s from p*
5.   if there exist ICs for (y,t)
6.     send (x,t)s according to ICs;
7.     wait until all influencees are done;
8.   end
9.   if there exist EOCs for (y,t)
10.    send (y,t) to the parent coordinator;
11.  end
12.  if there exist SCCs for (y,t)
13.    s := δ(x,s);
14.    M_prev := {M_i};
15.    {M_i} := ρ(s);
16.    for all processors p^j in {M_i}-M_prev
17.      send (query) to p^j;
18.      wait for (done,t_N)^j from p^j;
19.    end
20.  end
21.end
22.t_L := t;
23.t_N := min { t_N of all child processors };
24.send (done,t_N) to the parent coordinator;

Coordinator:when_rcv_(query)
1. send (query) to all child processors;
2. wait until all done
3. t_N := min { tN of all child processors };
4. send (done,tN) to the parent processor;
```

4 AGENTSIM SIMULATION ENVIRONMENT

This section illustrates a simulation environment for mobile agent systems, named ***AgentSim***, which implements the MDEVS formalism. AgentSim is implemented using the current development toolkit for mobile agent systems, IBM's Aglets Software Development Kit (ASDK) (Lange 1998).

4.1 Architecture of AgentSim

Figure 3 shows the overall architecture of the AgentSim simulation environment. The AgentSim consists of abstract simulators, a location manager and models created by a modeler. Each model created by a modeler includes the corresponding abstract simulator/coordinator, and becomes a mobile agent (See Figure 4). In the AgentSim environment, mobile agents representing each model conduct the simulation by collaboration.

A location manager is also an agent, and keeps the location information of every agents participating in the simulation. Mobile agents are capable of changing their execution environment. To communicate each other, they

must know their positions. If every agents look up the location manager whenever they try to communicate each other, the communication overhead will be large.

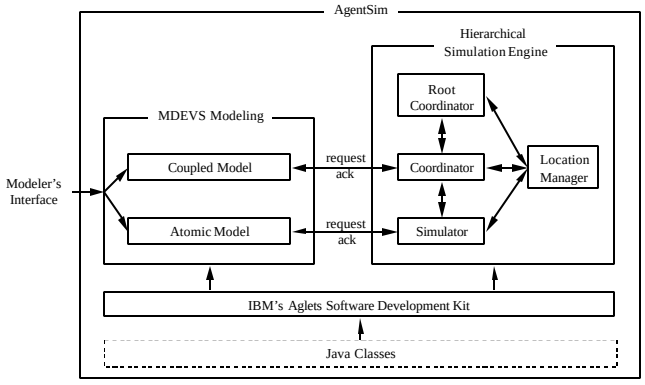

Figure 3: AgentSim Simulation Environment

For reducing communication overhead, each model agent has its own cache for the location information of other agents to which it sends the message. If the target agent moves to other execution environment, then the information of the local cache becomes invalid. ASDK provides the way of testing the validation of the information. Every agents look up the location manager only when the location information is invalid.

Every agent always reports the current location to the location manager whenever it is created or moved. However, the position of the location manager should not change during the simulation.

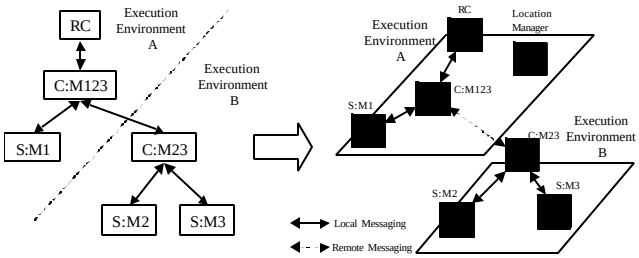

Figure 4: Mapping Models into Agents in AgentSim

4.2 Features of AgentSim

The AgentSim environment has the following features:

- **Fast Development of Mobile Agent Systems**; AgentSim implements each model as a real mobile agent. Each model physically sends, receives messages and changes its execution environment as the real mobile agent does. Thus, the models become the actual target system with the physical time instead of the simulation time (or virtual time). By this fashion, a user can rapidly develop the target mobile agent system.
- **Distributed Simulation Environment**; The mobile nature of the agent enables distributed simulation with AgentSim.
  - **Time Management Scheme**; AgentSim uses a synchronous time management scheme according to the abstract simulation algorithm of the MDEVS formalism. The root coordinator globally manages simulation time and advances it.
  - **Static/Dynamic Partitioning**; The mobile nature supports the dynamic partitioning (or dynamic load balancing) of processors. Before the simulation starts, a user can assign each model/agent to the specific execution environments, i.e., computers in the network.
  - **Location Transparency**; AgentSim provides the *Location Manager* that keeps the location information of each agent participating in simulation. Every agent can obtain the current location of the target agent by asking the name of the agent to the location manager. Therefore a user or a modeler doesn't need to know the physical address of the execution environments.

5 E-COMMERCE APPLICATION EXAMPLE

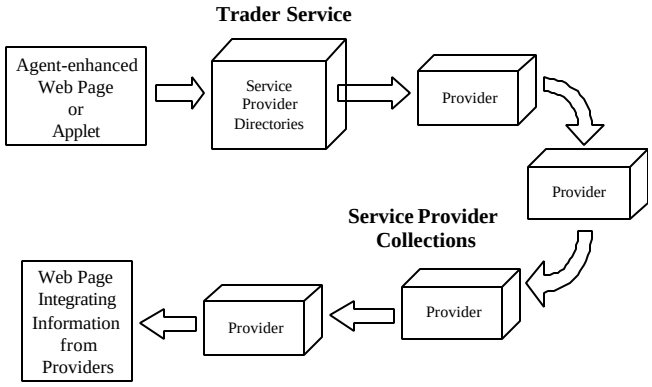

Figure 5: E-Commerce Example (Wong et al. 1999)

Figure 5 presents an example of an e-commerce application in which mobile agents negotiate with various providers for resources (Wong et al. 1999). Assume that an agent-congenial cyberspace is populated by providers of service for hosting agents and for allowing consumption of specific host resources. A user can delegate an agent to find information, buy things, or add value to information via Web page or specific user interface agent. A mobile agent created through user's request, might meet with a stationary travel agent at a Trader service to find appropriate processing services. With a potentially valid itinerary, the agent might visit any number of Service Providers, adding to its persistent knowledge store, making bids on goods and services. Along the way, the agent may further process information. The traveling agent performs data fusion and prepares its

information for presentation to an appropriate output media, such as a stationary avatar or a Web page.

Figure 6 depicts the model of the above e-commerce mobile agent system. **UserInterfaceAgent** representing a user creates an **Agent** by requesting to the **Host** model. The created **Agent** at the **Directory** finds an appropriate itinerary. According to the itinerary, the agent travels the **Providers** and performs the user's requests.

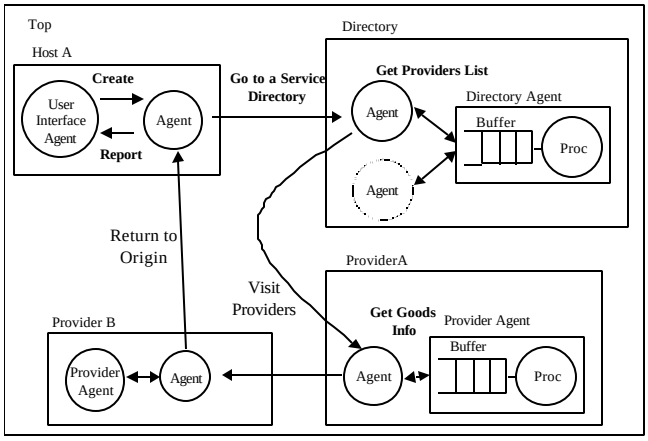

Figure 6: E-Commerce System Model

Figure 7 briefly describes the state transition diagram of the Agent.

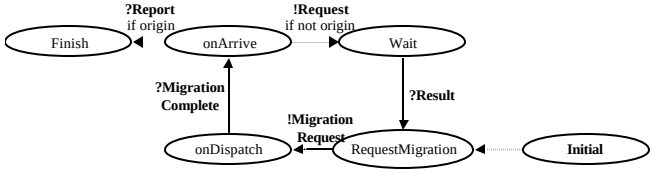

Figure 7: State Transition Diagram of **Agent**

6 CONCLUSION

The paper has presented the framework of the modeling and simulation for the mobile agent systems. In order to specify the mobile agent systems, we proposed the MDEVS formalism. The MDEVS formalism can specify the mobile agent systems by providing the formal methods capable of describing the structural changes of the system. Based on the MDEVS formalism, we developed simulation algorithms that are a conceptual basis of the implementation of AgentSim simulation environment. AgentSim is the simulation framework for mobile agent systems modeled by MDEVS formalism. An example of an e-commerce application demonstrates effectiveness of the proposed modeling and simulation framework.

REFERENCES

Barros, F. J. 1999. "DSDE: A Formalism For Representing Mobile Components." *Dagstuhl Seminar on Agent-Oriented Software Approaches in Distributed Modelling and Simulation*. Dagstuhl.

Cardelli, L. and A. D. Gordon. 1998. "Mobile Ambients." *In Proceedings of FoSSaCS, European Joint Conferences on Theory and Practice of Software*.

Franklin, S. and A. Graesser. 1996. "Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents." *In Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag.

Kiniry, J. and D. Zimmerman. 1997. "A Hands-on Look at Mobile Java Agents." *IEEE Internet Computing*, July-August, 21-30.

Lange, D. B. and M. Oshima. 1998. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley.

Stanski, P. and A. Zaslavsky. 1998. "Expressing Dynamics of Mobile Agent Systems Using Ambient Calculus." *In Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*. 434-439.

Uhrmacher, A. M. 1999. "JAMES – A Java-Based Agent Modeling Environment for Simulation." *Dagstuhl Seminar on Agent-Oriented Software Approaches in Distributed Modelling and Simulation*. Dagstuhl.

Wong, D., N. Paciorek and D. Moore. 1999. "Java-based Mobile Agents." *Communications of the ACM* 42, no. 3, 92-102.

Zeigler, B. P. 1984. *Multifacetted Modelling and Discrete Event Simulation*. Academic Press.

Zeigler B. P., T. G. Kim and C. Lee. 1991. "Variable structure modelling methodology: A dynamic computer architecture example." *Transactions of the Society for Computer Simulation* 7, no. 4, 291-319.

AUTHOR BIOGRAPHIES

Jae-Hyun Kim received the B. S. and M. S. degrees in Electrical Engineering in 1998 and 2000, respectively, from the Korea Advanced Institute of Science and Technology (KAIST), Taejon, Korea. His research interests include modeling, simulation, mobile agent and network management.

Tag Gon Kim is a Professor of Electrical Engineering & Computer Science, KAIST. His research interests include theory/practice of modeling/simulation, and simulation-based systems design methodologies. He is a senior member of IEEE and SCS, and a member of ACM and Eta Kappa Nu. He is an associate editor of *Simulation, Transactions of*

SCS, Simulation Digest. He is also an editor of *Int. J. in Intelligent Control and Systems*.