DEVS-Based Framework for Modeling/Simulation of Mobile Agent Systems

Jae-Hyun Kim and Tag Gon Kim
Department of Electrical Engineering & Computer Science
Korea Advanced Institute of Science and Technology (KAIST)
373-1 Kusong-dong Yusong-ku
Taejon, KOREA
E-mail: {jhkim, tkim}@smslab.kaist.ac.kr
Phone: +82-42-869-5454

This paper presents a 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 structural changes of the systems, which include the creation, addition, deletion, and migration of models and the dynamic changes of couplings between models. AgentSim is a software environment for the simulation and execution of mobile agent systems modeled by the MDEVS formalism. AgentSim is implemented as a library built on IBM’s Aglets. The modeling and simulation of an e-commerce mobile agent system is exemplified to demonstrate the effectiveness of the proposed modeling framework and the associated simulation environment.

Keywords: Mobile agent, mobile discrete event system specification, agent modeling/simulation, DEVS

1. Introduction
Mobile agents are software elements that can travel from one computer to another while executing their jobs. As a new network-programming paradigm, mobile agent systems will provide a convenient, efficient, and robust framework for implementing distributed applications. The benefits of the mobile agent systems are in addition to network aware applications, which can be extended to include internode mobility.

The building and debugging of mobile agent programs are more complex than those of ordinary distributed programs because the mobile agent systems require mobility while others do not. Thus, a formal method for developing and formulating the reasoning within a mobile agent system is required. The modeling and simulation of a mobile agent system during the development stage can significantly reduce development costs and time.

The mobile agent systems are categorized into discrete event systems with a capability for structural modification when viewed from the perspective that the system is an entire network within which mobile
agents are traveling (see Figure 1). Modeling the structural changes of the system requires the addition, deletion, and movement of models, as well as changes in the coupling relations between models. Conventional formalisms describing discrete event systems, such as DEV5 [9], Petri-Net [15], CCS [12-13], and CSP [14], do not support dynamic changes of the structure. It is thus very hard to model the mobile agent system itself. Methodologies for the representation and simulation of variant model families and variable structure models also have been proposed [10, 17]. They focus on representing systems that are able to undergo structural change including the addition and deletion of systems and the modification of relations among components, with the exception of mobility within the model. Recent research that describes mobile agents and/or objects based on process algebras, Petri-Nets, coordination languages, temporal logics, and other studies ([1, 2, 7, 11]) which focus on the expression of mobility are not sufficient [11]. The mobility of a composite mobile agent or a group of mobile agents should be considered as well as that of a single agent. An agent is not always a single and isolated entity that migrates and executes alone. An agent may possibly consist of a group of component agents, which act and move together. A formalism should be able to model group and interagent migration to sufficiently express the mobility of mobile agents.

A rapid prototyping of the target system is a hot issue in many disciplines. There is always a gap between physical mobile agents and their models. A real mobile agent is directly transported to other execution environments over the network, but most previous
models only traveled within the model hierarchy. We have tried to reduce this gap to facilitate the rapid development of mobile agent systems. Figure 2 describes the overall framework. As shown, this framework covers the entire procedure of developing mobile agent systems. A modeler uses the mobile discrete event system specification (MDEVS) formalism in order to specify and model the target mobile agent system. The MDEVS formalism is an extension of the DEVS formalism [9] and supports the specification of the mobile agents in a hierarchical, modular manner. We have developed an environment called AgentSim, which supports simulation and execution of the mobile agent models. AgentSim is implemented as a library for IBM's Aglets [5]. With the simulation engine of the AgentSim, a modeler can simulate the mobile agent models to validate and verify the model. Being a distributed simulation environment by nature, AgentSim can simulate the large-scale network models, which are examples of mobile agent applications. Each model in AgentSim is implemented for actual mobile agents and already has the basic functionality of mobile agents. In this fashion, modelers can prototype the target systems rapidly.


This section proposes the Mobile Discrete Event System Specification (MDEVS) formalism for the modeling, simulation, and analysis of mobile agent systems. The MDEVS formalism is an extension of the DEVS formalism [9].

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

MDEVS has two basic models—these are the atomic and coupled models. An atomic model represents the behavior of the indivisible component of the system, whereas a coupled model is a model which contains other models inside. The coupled model can have not only atomic but also coupled models because of the closed under coupling property. A coupling specification describes how components are interconnected to form the desired coupled model. Simulation messages are delivered to the destination model according to that coupling specification.

A mobile agent can be described as a single atomic model or a coupled one. It is preferable to construct a mobile application by organizing more than one large and monolithic mobile agent in many cases. A mobile agent can be divided into many small components and each component can be represented as an atomic/coupled model. This hierarchical development of the system is essential to large-scale mobile applications.

2.1 Atomic Model

An atomic model has specifications for the dynamics of the model. It describes the behavior of a component, which is indivisible, at a timed state transition level. An atomic model of the proposed MDEVS formalism is the same as the original one. Formally, the following 7-tuple [9] specifies an atomic model AM:

\[ AM = < X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta > \]

with the following constraints:

\[ X : \text{Input Events Set}; \]
\[ Y : \text{Output Events Set}; \]
\[ S : \text{States Set}; \]
\[ \delta_{ext} : Q \times X \rightarrow S, \text{External Transition Function}; \]
\[ \delta_{int} : Q \rightarrow S, \text{Internal Transition Function}, \lambda : Q \rightarrow Y, \text{Output Function}, \]
\[ ta : S \rightarrow R^+ \cup \{-, \}, \text{Time Advance Function}. \]

As with modular specifications in general, we must view the above atomic DEVS model as possessing input and output ports through which all interactions with the external world are mediated. To be more specific, when all external input events are arriving from other models and received on its input ports, the model decides how to respond to them by its external transition function. In addition, when no external events arrive until the scheduled time, which is specified by the time advance function, the model changes its state by the internal transition function and reveals itself as external output events on the output ports to be transmitted to other models.

2.2 Coupled Model

A coupled model joins several components 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.

Unlike a coupled model defined in the DEVS formalism, a coupled model of MDEVS formalism is capable of changing the inside structure. Changing structure generally means that the subordinate models can be newly added, deleted and moved, and the relation between models, i.e., coupling, can be dynamically altered. Formally, the following 8-tuple specifies a coupled model CM:

\[ CM = < X, Y, S, \rho, \delta, [M_i], C, \text{SELECT} > \]

with the following constraints:

\[ X = X_{in} \cup X_{chr}, \]
\[ X_{in} : \text{Input Events Set}, \]

\[ X_{chr} : \text{Communication Events Set}, \]
\[ X_{out} : \text{Output Events Set}, \]
\[ X_{sel} : \text{Selection Events Set}, \]
\[ X_{chan} : \text{Channel Events Set}. \]
$X_{ch}$ : Structure-Change Events Set,
$Y$ : Output Events Set,
$S$ : Structure-States Set;
$[M_i] \in M^*$ : Activated Models Set,
where $M^*$ : Total Models Set,
$\rho : S \rightarrow 2^{M^*}$, Model Activation Function,
$\delta : X_{ch} \times S \rightarrow S$, Structure Transition Function,
$C = \{EIC, EOC, IC, SCC\}$: Couplings Set,
$\text{EIC} \subseteq X_{in} \times \bigcup_i X_i \times S$, External Input Coupling Relation,
$\text{EOC} \subseteq \bigcup_i Y_i \times Y \times S$, External Output Coupling Relation,
$\text{IC} \subseteq \bigcup_i Y_i \times \bigcup_i X_i \times S$, Internal Coupling Relation,
$\text{SCC} \subseteq \bigcup_i Y_i \times X_{ch} \times S$, Change Coupling Relation,
$\text{SELECT} : 2^{[M_i]} - \phi \rightarrow M_i$, 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 determines the activated model's set and the activated couplings. A received structure-change event alters 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, it is 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.** [9] A formalism is said to be closed under coupling if any composite systems obtained by coupling components specified by the formalism are themselves 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.

---

**Example 1.** This example describes how to model the systems with MDEVS formalism. Figure 3 shows the example system. Model A is moving from model D to model E by sending an event through output port. D deletes A when it receives the MoveOut event, and E adds A when it receives the MoveIn event. Note that the state of A should be invariant during a migration process. Models D and E are specified as follows:

$$D = \{X, Y, S, [M_i], \rho, \delta, C, \text{SELECT}\}$$

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

$$Y = \{\text{out}\};$$

$$S = \{A_{IN}, A_{OUT}\};$$

$$\rho(A_{IN}) = \{A, B\}, \rho(A_{OUT}) = \{B\};$$

$$\delta(\text{MoveOut}, A_{IN}) = A_{OUT};$$

$$\text{EOC} = \{(A, \text{out}, out, A_{IN})\};$$

$$\text{SCC} = \{(A, \text{out}, \text{MoveOut}, A_{IN})\};$$

$$\text{SELECT} \quad \{A, B\} = A.$$

$$E = \{X, Y, S, [M_i], \rho, \delta, C, \text{SELECT}\};$$

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

$$S = \{A_{IN}, A_{OUT}\};$$

$$\rho(A_{IN}) = \{A, C\}, \rho(A_{OUT}) = \{C\};$$

$$\delta(\text{MoveIn}, A_{OUT}) = A_{IN};$$

$$\text{IC} = \{(A, \text{in}, A_{IN}), (C, \text{out}, A_{IN})\};$$

$$\text{SELECT} \quad \{A, C\} = A.$$

---

**Table 1. Simulation Messages**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Send</th>
<th>Receive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(*)</td>
<td>Star</td>
<td>RC,C</td>
<td>C,S</td>
<td>Internal event(*) at time t</td>
</tr>
<tr>
<td>(x,t)</td>
<td>X</td>
<td>C</td>
<td>C,S</td>
<td>External event(x) at time t</td>
</tr>
<tr>
<td>(y,t)</td>
<td>Y</td>
<td>C,S</td>
<td>C</td>
<td>Output(y) at time t</td>
</tr>
<tr>
<td>(done,t_n)</td>
<td>Done</td>
<td>C,S</td>
<td>RC,C</td>
<td>Event handling is done, and next scheduling time is t_n</td>
</tr>
<tr>
<td>(query)</td>
<td>Query</td>
<td>RC,C</td>
<td>C,S</td>
<td>Query the next scheduling time</td>
</tr>
</tbody>
</table>

S : Simulator, C : Coordinator, RC : Root Coordinator
Table 2. Event Handling Algorithms for Abstract Simulator

```
Simulator: when_rcv_(x,t)
  t = t�� - t身;
  1. e := t - t身;
  2. s := δ בהם(s,e,x);
  3. t身 := t身;
  4. t身 := t身 + ta(s);
  5. send (done,t身) to the parent coordinator;

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

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

3. Abstract Simulation Algorithm

The MDEVS formalism uses the abstract simulator concept for simulation. Zeigler proposed the abstract simulator concept [9] 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.

There are two types of processors—a simulator for an atomic model and a coordinator for a coupled model. A special kind of coordinator called a root coordinator is not associated with any models and takes responsibility for advancing the 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 originate from the abstract simulation algorithm of DEVS formalism. The query message is added in order to update the 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 in a timely. A simulator for an atomic model is depicted in Figure 4. It receives and processes x message (x, t)s, star message (*, t), and query messages. As a result, it produces y message (y, t)t and done message (done, tN)t. Because a simulator associates to an atomic model, which is always a leaf node of a system decomposition tree, it cannot receive (y, t) and (done, tN)t messages. The event handling algorithms for the abstract simulator are presented in Table 2.
Table 3. Event Handling Algorithms for Coordinator

```
Coordinator: when r_cv_ (x,t)
  t = t_x
  1. if x ∈ X
     2. for each affected processor p'
        3. send (x,t) to p'
        4. wait for (done,t) from p'
     5. end
  6. else
     7. s := δ(x,s);
     8. M_{r_cv} := M_j;
     9. M_i := p(s);
    10. for all processors p' in (M_i) - M_{r_cv}
       11. send (query) to p'
       12. wait for (done,t) from p'
    13. end
    14. end
    15. t_x := t;
    16. t_x := min { t_x of all child processors }
    17. send (done,t) to the parent coordinator;

Coordinator: when r_cv_ (r,t)
  t = t_y
  1. find the imminent processors with min t_x
  2. select one, p', and send (r,t) to it
  3. wait for (done,t) from p'
  4. for all received (y,t) from p'
     5. if there exist ICs for (y,t)
        6. send (x,t) according to ICs
        7. wait until all influences 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_{r_cv} := M_j;
       15. M_i := p(s);
       16. for all processors p' in (M_i) - M_{r_cv}
          17. send (query) to p'
          18. wait for (done,t) from p'
       19. end
    20. end
    21. end
    22. t_x := t;
    23. t_x := min { t_x of all child processors }
    24. send (done,t) to the parent coordinator;

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

3.2 Coordinator for Coupled Models

The responsibility of a coordinator for a coupled model is to synchronize the component abstract simulators for scheduling the next event time and routing 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 structural modification.

A coordinator for a coupled model is depicted in Figure 5. It receives and processes x message (x, t)x, star message (*, t)s, done message (done, t_N)s, and query messages. As a result, it produces y message (y, t)s and done message (done, t_N)s. The event handling algorithms for the coordinator are presented in Table 3.

Example 2. To demonstrate the proposed simulation algorithm, we revisit the example in Figure 3. Assume that the next scheduling time t_N of model A is the earliest one among all atomic models, and model A sends an event to two coupled models, D and E through the Out port. Model D deletes model A, and
Model A moves from D to E, by sending an output event through `out` port.

1. RC sends `(*,t)` to S:A which has the minimal \( t_N \).
2. S:A sends `(y,t)` to C:D.
3. S:A finishes `(*,t)` handling and it sends `(done, t_N)` to C:D.
4. C:D forwards received `(y,t)` to C:F.
5. **C:D removes S:A from Activated Models Set**
6. C:D sends `(done, t_N)` to C:F
7. C:F translates `(y,t)` to `(x,t)` and sends it to C:E.
8. **C:E add S:A to the Activated Models Set.**
9. C:E sends `(query)` to S:A.
10. S:A sends `(done, t_N)` to C:E.
11. C:E sends `(done, t_N)` to C:F.
12. C:F sends `(done,t_N)` to RC.

**Figure 6.** An Example of Hierarchical Simulation

**Figure 7.** Architecture of AgentSim
model E adds model A when each receives the event. Note that the states of model A are preserved during movement. Model A can move from D to E in this fashion. Figure 6 shows the sequence of the simulation message of this example.

4. AgentSim Environment

AgentSim is a software environment for simulation and execution of the MDEVS models. AgentSim is implemented as a library built on Aglets [5]. Aglet is a Java-based mobile agent, which resides in Aglet servers in the network. Each MDEVS model is developed as an individual Aglet, which processes specified simulation messages. A modeler can build a new model by implementing message handling functions using the AgentSim library.

Figure 7 shows the overall architecture of the AgentSim environment. There are two different engines in AgentSim—simulation and execution engines. The simulation engine consists of the abstract simulator/coordinator and a location manager. Each model created by a modeler includes the corresponding abstract simulator/coordinator, and becomes a mobile agent. A location manager is also an agent, and keeps the location information of every agent participating in the simulation.

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 another execution environment, then information in the local cache becomes invalid. The Aglets Software Development Kit (ASDK) [5] provides the way of testing the validation of the information. When the agent tries to send a message to the target, it can notice that the information in the local cache is invalid, and looks up the new location information by asking the location manager and updating the local cache. Every agent always reports the current location to the location manager whenever it is created, deleted or moved to another execution environment, i.e., Aglets servers in the network. The position of the location manager should not change during the simulation.

Simulation is the execution of models according to the associated simulation algorithm. Each simulator and coordinator advances simulation time through the communicating process. In the AgentSim environment, mobile agents representing each model conduct the simulation by sending and receiving simulation messages (see Figure 8).

The structure of the execution engine is the same as that of the simulator. The only difference is that the execution engine uses the physical time instead of the simulation virtual time. When simulating the target system, the behavior of each model is almost the same as the real one. Each model physically sends and/or receives messages and even can change its execution environment as the real mobile agent does. Therefore, if the simulation time (or virtual time) is replaced by the physical time, then the models become the actual target system. In this fashion, a user can rapidly develop the target mobile agent system.

5. E-Commerce Application Example

This section provides an example of modeling and simulation of a mobile agent system using the MDEVS formalism. Figure 9 presents an example of an e-commerce application in which mobile agents negotiate with various providers for resources [8]. The MDEVS models of the e-commerce system will be presented. Simulation results will also be presented to verify the correctness of the behavior of the system and to evaluate performance indices.

Figure 10 depicts the model of the above e-commerce mobile agent system. Assume that a user com-
municating with a user interface agent in Host A wants to buy a book. He creates an agent by commanding with a User Interface Agent and delegates to the agent a job to find a good book using the Internet bookstores. The agent first visits the Service Directory to find the address of the registered bookstores. The Directory Agent resident in the Service Directory keeps the information of providers in its database. After getting the list of bookstores, i.e., Provider A and Provider B, the agent begins to visit providers. In each provider site, a Provider Agent is waiting for customers. By communication with the Provider Agent, our agent can obtain a list of books, or even purchase a book.

The typical behavior of mobile agents is simple. They do almost the same things when traveling the network. They arrive at a host, do their job, and leave. The state transition of our agent model is depicted in Figure 11. The agent model does its job by requesting and receiving information. After it finishes its job, it requests the migration to the site manager, and the migration is done when the Migration Complete event arrives.

Figure 12 shows the initial partition of the models, the agents in each environment and the screen shots of the Aglet servers. The screen shot shows two Aglet viewers called Tahiti, which displays a list of aglets, currently residing in each server. Figure 13 describes the state transition diagram of the Agent models created in the Host A and B. Note that the Agent model created in the Host A travels to Directory, Provider A, and Provider B, while the Agent model created in the Host B goes to Directory and Provider C. This result shows the correctness of the behavior of the Agent model. Figure 13 also shows the timing difference between a simulation and an execution. We replaced the simulation engine with the execution engine, and measured the starting time of the state transition. The results that we obtained showed that the state transition of the Agent model during the execution occurred an average of 104.2 msec late when compared to the simulation time. This is because the execution engine of AgentSim is not a hard real time engine, but this is negligible in this application because the users cannot notice a 0.1 sec delay of each state transition of the agent.

Now we evaluate the sample performance of the E-commerce application system. We will examine the scalability of the example E-commerce system by measuring the average service time of the Agent at the Directory or the Provider and the time taken for an agent to complete its job. Scalability would be an important factor in developing the E-commerce system because the system performance should not decrease as the number of users increases.

Table 4 summarizes the parameters of the system. The number of hosts, i.e., the number of users, increases from 30 to 100, while the number of the Directory and the providers registered in the Directory remain constant. Each host sends the agents to the Directory in the given interval. The processing time of a request from an agent in the Directory or the Provider is given in Table 4.

Figure 14 shows the average service time of the Directory and the Provider. Note that as the number of hosts increases and the number of agents increases, each agent waits a longer time to be served. The Directory especially becomes a bottleneck. As a result, each agent takes a longer time to complete its job as more users navigate the system. Figure 15 shows this result. From this performance evaluation, we conclude that it is desirable to use, not a centralized Directory, but many distributed Directories in order to maintain the service quality.
Figure 11. State Transition Diagram of Agent

Figure 12. Initial Partition of Models
6. Conclusion
This paper has presented the framework for developing mobile agent systems. Mobile agent technology is an emerging technology, but poorly supported by existing formal methods. Mobile agents in the category of discrete event systems are especially capable of structure transformation. In order to specify the mobile agent systems, we proposed the MDEVS formalism. The MDEVS formalism can specify the mobile agent systems by providing formal methods capable of describing the structural changes within the system.

Based on the MDEVS formalism, we also proposed the abstract simulation algorithm, which is a conceptual basis for the implementation of the AgentSim environment. AgentSim is the simulation and execution environment for mobile agent systems modeled by MDEVS formalism. Implemented on top of Aglet, AgentSim facilitates the rapid development of mobile agent systems as well as the timing and behavioral verification of the model through simulation. It also supports large-scale simulation in a distributed fashion. Finally, we provided an e-commerce application as an example of our modeling and simulation framework.

7. Appendix
Proof of Theorem 1. The association mapping CM into AM demonstrates the closure of the formalism under coupling. We will describe

\[ AM = \langle X_A, Y_A, S_A, \delta_{ext}, \delta_{int}, \lambda, ta \rangle \]

in terms of the elements in
First, the input/output events set and the states set of AM are defined as follows:

\[
X_A = X_C, \quad Y_A = Y_C, \quad S_A = (S_i) \times S_C
\]

where \(S_i\) is a state set of \(M_i \in \{M_i\} = \rho(s_C)\), and \(s_C \in S_C\). We define \(\sigma_i\) by \(\sigma_i = t_a(s_i) - \varepsilon_i\), where \(t_a\) is a time advance function of \(M_i\), \(s_i \in S_i\) is the current state of \(M_i\), and \(\varepsilon_i\) is an elapsed time of \(M_i\). Then, the time advance function of AM is defined by \(t_a(s) = \min \{\sigma_i \text{ for all } M_i \in \{M_i\} = \rho(s_C)\}\). To define the transition functions, let the current state \(s \in S_A\) become the new state \(s' \in S_A\) by transition. By the external transition function \(s' = \delta_x(s, e, x)\) each component of the state \(s\) is transited to the new component as follows:

\[
s'_c = \begin{cases} 
\delta(s_c, x) & \text{if } x \in X_{ch} \\
\emptyset & \text{otherwise} 
\end{cases}
\]

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 \(\rho(s_C) = \{M_i\}\) and \(\rho(s'_C) = \{M'_i\}\).

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

\[
s'_c = \begin{cases} 
\delta(s_c, x) & \text{if } (y_i, x, s_C) \in EIC, y_i = \lambda_i(s_i), \sigma_i = 0 \\
\emptyset & \text{otherwise} 
\end{cases}
\]

The output function is defined as

\[
y = \lambda(s) = \lambda_i(s_i) \text{ if } (y_i, y, s_C) \in EOC, M_i \in \rho(s_C) .
\]

8. References


Jae Hyun Kim received a BS and MS in electrical engineering in 1998 and 2000, respectively, from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. Currently, he is a PhD student at the Department of Electrical Engineering and Computer Science, KAIST. His research interests include methodology for modeling and simulation of discrete event systems and mobile agent systems. He is a student member of IEEE.

Tag Gon Kim received his PhD in computer engineering with specialization in methodology for systems modeling/simulation from the University of Arizona, Tucson, AZ, 1988. He was a full-time instructor at the Communication Engineering Department of Bukyung National University, Busan, Korea between 1980 and 1983, and an assistant professor at the Electrical and Computer Engineering Department at the University of Kansas, Lawrence, Kansas, USA from 1989 to 1991. He joined the Electrical Engineering Department of KAIST, Daejeon, Korea in Fall, 1991 as an assistant professor and has been a full professor since Fall 1998. His research interests include methodological aspects of systems modeling simulation, analysis of computer/communication networks, and development of simulation environments. He has published more than 100 papers on systems modeling, simulation and analysis in international journals and conference proceedings. He is a co-author (with B.P. Zeigler and H. Prachofer) of the book "Theory of Modeling and Simulation" (2nd ed.), Academic Press, 2000. He is the Editor-in-Chief of Transactions of The Society for Modeling and Simulation published by The Society for Modeling and Simulation International (SCS). He is a senior member of IEEE and SCS and a member of ACM and Eta Kappa Nu.