Proceedings of 2001 Seoul International Simulation Conference, pp.453-458, October, 2001, Seoul, Korea.

Methodology for Discrete Event Modeling/Simulation of Mobile Agent Systems

Jae-Hyun Kim and Tag Gon Kim

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

ABSTRACT

A mobile agent is an autonomous software agent capable of moving from one computer to another while performing its tasks. We view the agent as a discrete event system in the view of its computation and communication. This paper presents a methodology for modeling and simulation of such a mobile agent system as a discrete event system. The methodology is based on the Mobile Discrete Event System Specification (MDEVS) formalism and the associated simulation environment AgentSim which are previously developed by the authors. Within the methodology an atomic model represents dynamics of a mobile agent; a coupled model is modeled as mobile agent servers for representation of structural changes between atomic agents. Being based on the object-oriented environment the modeling methodology exploits inheritance of basic classes *AtomicModel* and *CoupledModel* provided by AgentSim.

1 INTRODUCTION

A mobile agent is a software autonomous agent with mobility. It is a piece of software that is designed to be able to travel from one computer to another while executing its jobs. A host computer provides a mobile agent server in which a mobile agent can reside and use resources of the computer. A mobile agent system consists of mobile agent servers and mobile agents.

Mobile agent systems effectively reduce network traffic, and help people to construct more robust and fault-tolerant applications through their ability to operate asynchronously and autonomously of the process that created them[5].

The building and debugging of mobile agent programs are more complex than that 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 in the development stage can significantly reduce development costs and time.

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. Therefore discrete event modeling and simulation methodology is applied to mobile agent systems.

A formal modeling and simulation framework for mobile agent systems has already been proposed recently[1][2]. For modeling of mobile agent systems, Mobile Discrete Event System Specification (MDEVS) formalism has been proposed. MDEVS formalism is able to represent the dynamics of mobile agent systems including movement and dynamic coupling. For simulation of mobile agent systems, AgentSim is provided as a library, which facilitates various functions required to simulate MDEVS models.

This paper presents methodology for discrete event modeling and simulation of mobile agent systems, and focuses on how to model and simulate mobile agent systems within the framework.

Section 2 briefly introduces the framework for modeling/simulation of mobile agent systems. In Section 3, we mention how to model mobile agent systems using MDEVS formalism. Section 4 describes how to implement MDEVS models with AgentSim and how to simulate them. Section 5 concludes the paper.

2 OVERVIEW OF THE FRAMEWORK

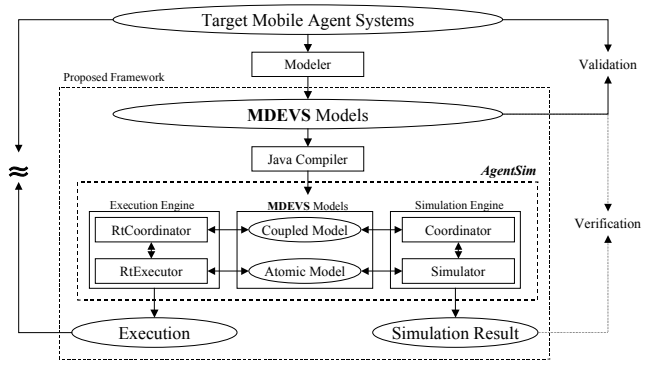


Figure 1 The Framework for Modeling/Simulation of Mobile Agent Systems. [2]

Figure 1 shows the framework for modeling and simulation of mobile agent systems introduced in [2]. The framework covers the entire procedure of modeling and simulation of mobile agent systems. A modeler uses MDEVS formalism to specify the target mobile agent system. The MDEVS formalism is an extension of the DEVS formalism[3] and supports the specification of the mobile agents in a hierarchical, modular manner. The MDEVS formalism is able to express structural changes within the systems, which include the creation, addition, deletion, and migration of models and the dynamic changes of couplings between models.

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[4]. With the simulation engine of the AgentSim, a modeler can simulate the mobile agent models to validate and verify the model. As a distributed simulation environment by nature, AgentSim can simulate the large-scale network models, which are examples of mobile agent applications.

3 MODELING OF MOBILE AGENT SYSTEMS

This section shows how to model mobile agent systems using MDEVS formalism. Based on the basic behavior presented in Section 3.1, an example model of a mobile agent is presented. Modeling of mobile agent servers using the coupled model is described in Section 3.3.

3.1 Behavioral Patterns of Mobile Agents

The concept of a mobile agent is now emerging and mobile agent applications are now being pioneered. In this first stage of the new technology, some behavior patterns of mobile agents are found and useful design patterns are suggested[5].

Although the specific behavior of mobile agents may differ depending on their missions that they get, the very basic behavior is always the same; *Arrival-Computation-Departure*. Most mobile agents move to the destination host, and perform their mission using the computational power of the host, and then leave to another destination. After repeating this pattern until the agents achieve their goal, they usually return home to report the result to the commander.

The whole life cycle is described as follows:

- A user or an agent creates mobile agents.
- A user or an agent delegates a job to the created agent.
- The agent moves to the designated host.
- The agent uses resources or interacts with other agents in that host.
- The agent moves to another host and continues its job.
- If the agent completes the job, it returns home and reports the result to the user or the creator.

3.2 Modeling of Mobile Agents

Mobile agents are usually a single piece of software. It is recommended that a mobile agent is modeled as an atomic model because the agent is a single indivisible component in the discrete event system.

Making an atomic model is specifying 7 elements below:

$$\mathbf{AM = < X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta >[1][2][3]}$$

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

Figure 2 shows an example state transition diagram based on the basic behavior in section 3.1. The *Computing* state aggregates specific states required to do the delegated job. This figure depicts only the basic pattern of *Arrival-Computation-Departure***.** Adding detailed behavior for the mission in this example makes the mission-specific mobile agent model.

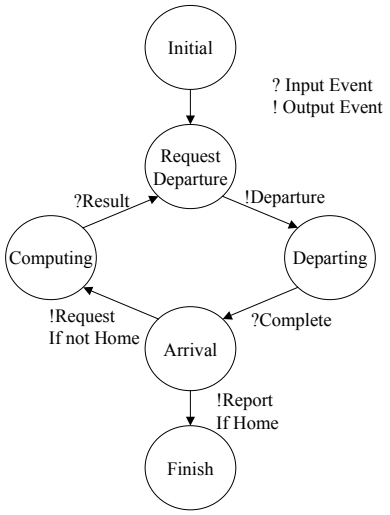
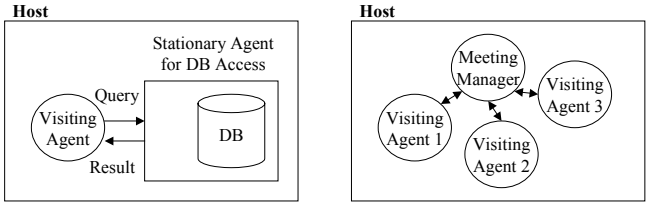


Figure 2 State Transition Diagram of a Mobile Agent

3.3 Modeling of Mobile Agent Servers

A mobile agent server in the network provides computational power and resources for mobile agents inside. A server can allow direct access to resources such as hard disks, memory or other programs. However, due to security problems, a server may provide indirect ways to access resources or may prevent it from accessing specific resources. One method for indirect access is using a stationary agent through which the visiting agents can safely access resources (Figure 3). Also the agent server is a place where agents can meet each other and exchange information.



(a) DB Access through Stationary Agent (b) Interaction with other agents

Figure 3 Modeling of Mobile Agent Servers

For the server provides various kinds of services and agents arrive to and depart from the server, a coupled model is suitable for modeling a mobile agent server. The services can be modeled as a component in the server model. The MDEVS coupled model is defined as follows.

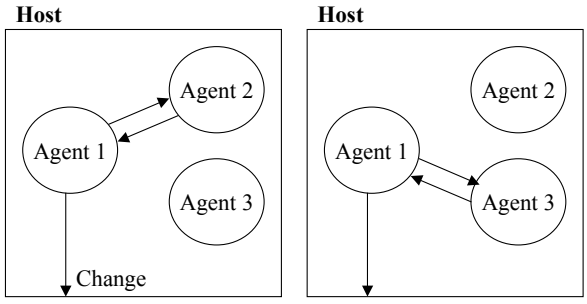
$CM = \langle X, Y, S, \rho, \delta, \{M_i\}, C, SELECT \rangle [1][2]$

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;
$\rho : S \rightarrow 2^{M^*}$, Model Activation Function;
$\delta : X_{ch} \times S \rightarrow S$, Structure Transition Function;
C = { EIC, EOC, IC, SCC }: Couplings Set;
$EIC \subseteq X_{in} \times \bigcup_i X_i \times S$, External Input Coupling Relation;
$EOC \subseteq \bigcup_i Y_i \times Y \times S$, External Output Coupling Relation;
$IC \subseteq \bigcup_i Y_i \times \bigcup_j X_j \times S$, Internal Coupling Relation;
$SCC \subseteq \bigcup_i Y_i \times X_{ch} \times S$, Structure-Change Coupling Relation;
SELECT : $2^{\{M_i\}} - \phi \rightarrow M_i$, Select Function.

The coupled model has structure-state, structure transition function, and structure-change coupling in order to express dynamic coupling and model movement. Structure-state represents current models and their couplings. The change of structure-state means the change of inside models and couplings. The change of a structure-state occurs by structure transition function when the coupled model receives a structure-change event through a structure-change coupling.

Figure 4 shows an example of dynamic coupling. Initially Agent 1 is coupled with Agent 2. Agent 1 sends a structure-change event *Change* through structure-change coupling between Agent 1 and Host. The coupled model Host receives the event and changes the structure-state by structure transition function. The new structure-state represents the second Host model as shown in Figure 4 (b).



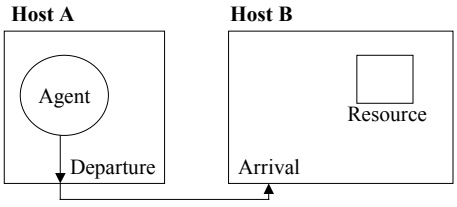
(a) Agent 1 requests for Host to change its coupling

(b) Host deletes couplings between Agent 1 and 2, creates couplings between 1 and 3

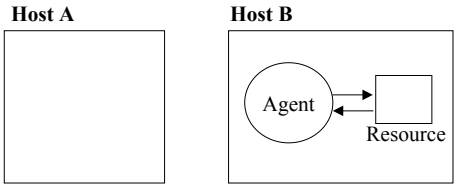
Figure 4 Dynamic Coupling

Similar to this procedure, the migration of a mobile agent between two mobile agent servers is represented. Figure 5 depicts the migration sequence. The Agent sends

the structure-change event, *Departure*, to Host A and *Arrival* to Host B at the same time. Host A deletes the Agent and Host B adds the Agent. The associated couplings in Host A are also deleted and new couplings are added in Host B.



(a) Agent requests Host A for departure to Host B; the event should be delivered to both Host A and B



(b) Host A deletes Agent and associated couplings; Host B adds Agent and requested couplings

Figure 5 Migration of Agent from Host A to B

4 IMPLEMENTATION & SIMULATION

After you have finished modeling in MDEVS formalism, implementation using AgentSim APIs and compilation are required to perform the simulation.

4.1 Implementation

As a simulation environment for mobile agent systems, we have developed AgentSim. It is a library based on IBM's Aglets. Aglet is a java-based mobile agent system development toolkit. The reason why we do not develop a C/C++ simulator and choose a java-based mobile agent development toolkit as a simulation environment is that modeling and simulation is one step for development. Modeling and simulation on top of mobile agent systems may reduce the gap between the model and the target system; hence the rapid development of the system is possible[2].

Derived from the *Aglet* class, we built the *Model* class that became a parent class of the *AtomicModel* and *CoupledModel* class. Modelers are able to create their own models by deriving a new class from the *AtomicModel* or *CoupledModel* class. Basic functions for manipulating tuples of an Atomic or Coupled model are provided (Table 1,Table 2).

Table 1 AgentSim API for Atomic Model

Atomic Model	Function
X	`addInport("Input");` `msg.get("Port");`
Y	`addOutport("Output");` `msg.put("Port",Object);`
S	`StateVar.put("State","Initial");` `StateVar.get("State");`
δ_{ext}	`void ExtTransFn(Hashtable stateVar, double timeE, Messages msg);`
δ_{int}	`void IntTransFn(Hashtable stateVar)`
λ	`void OutputFn(Hashtable stateVar, Messages msg)`
ta	`double TimeAdvanceFn(Hashtable stateVar)`

Table 2 AgentSim API for Coupled Model

Coupled Model	Function
X	`addInport("Inport");`
Y	`addOutport("Output");`
S	`StateVar.put("State","Initial");` `StateVar.get("State");`

ρ	`createChild(model,"classname",arg);`
	`addChild(model);`
	`deleteChild(model);`

δ	`void StateTransFn(Hashtable stateVar, Messages message)`
C	`addCoupling(src,"sport",dst,"dport");` `deleteCoupling(model,"port");`

Figure 6 and Figure 7 show the example code for *Agent* and *Agent Server*. *Agent* is modeled and implemented as an atomic model. The basic implementation of an atomic model is to override 4 characteristic functions of atomic model, i.e., ExtTransFn, IntTransFn, OutputFn, TimeAdvanceFn. To initialize the model, the AgentSim provides *onCreation* function.

AgentServer class is an example implementation of a coupled model. The *onCreation* function should include the creation of input / output ports, models, couplings between models, and the creation and initialization of structure state. The *StateTransFn* function handles state-change events delivered to the model.

```
class Agent extends AtomicModel {

 void onCreation(Object o)
 {
  // Called when the Agent is created.
  // Create input/output ports
  // Initialize State
  addOutport("Request");
  StateVar.put("State","Initial");
  ...
 }

 void ExtTransFn(Hashtable stateVar, double timeE,
Messages msg)
 {
```

```
  // Handle External events
  if( stateVar.get("State").equals("onDispatch")
      && msg.getPort().equals("MoveOutDone") )
  {
   stateVar.put("State","onArrive");
  }
  else ...
 }

 void IntTransFn(Hashtable stateVar)
 {
  // Handle Internal events
  if( stateVar.get("State").equals("Initial") )
  {
   stateVar.put("State","makeConnect");
  }
  else ...
 }

 void OutputFn(Hashtable stateVar, Messages msg)
 {
  // Prepare output events and
  // send them through output ports
  if( stateVar.get("State").equals("onArrive")
     && Host.equals(Origin) )
  {
  // Report results
   msg.setPortVal("Report",result);
  }
  else ...
 }

 double TimeAdvanceFn(Hashtable stateVar)
 {
  // return time

if(  ((String)StateVar.get("State")).equals("Initia
l") )
   return 10;
  else ...
 }
}
```

Figure 6 Example Code for Agent

```
class AgentServer extends CoupledModel {

 void onCreation(Object o)
 {
  // Called when the Agent is created.
  // Create input/output ports
  // Initialize Structure state
  addOutport("MoveOutRequest");
  StateVar.put("State","Wait");

  // create inside models
  createChild(directory,"classname",arg);

  // add couplings between models
  addCoupling(directory,"connect",
     thismodel,"ConnectRequest");
  ...
 }

 void  StateTransFn(Hashtable  stateVar,  Messages
message)
 {
  // Handle State-change events
  if( message.getPort().equals("MoveIn") )
  {
   // Transition of Structure-state
   StateVar.put(moveInModel,"In");
   // Add new child model
   addChild(moveInModel,moveInModel,"In");
   // Add new coupling
   addCoupling(moveInModel,"Request",
      directory,"in",moveInModel,"In");
   // delete coupling
   deleteCoupling(directory,"out");
   // delete model
   deleteChild(model,false,model,"Out");
  }
  else ...
 }

}
```

Figure 7 Example Code for Agent Server

4.2 Simulation

Like other Aglets, each simulation model of the compiled form runs in Aglet servers. After uploading all models and simulation control agents in appropriate Aglet servers in the network, the simulation is ready to run. The partitioning of the models among the servers is manually done. The graphical viewer *Tahiti* of the agent server enables us to check the current status of the agents[4].

5 CONCLUSION

In this paper we have presented a methodology for discrete event modeling and simulation of mobile agent systems using the developed framework.

We examined the basic pattern of behavior of a mobile agent, and from that pattern we have suggested the basic model of a mobile agent. However, modeling of the target systems totally depends on modelers. This paper only shows one possible modeling method. You can even model a big complex mobile agent with a coupled model.

We hope that the methodology and framework for modeling and simulation of mobile agent systems will help developers of the systems and applications.

REFERENCE

[1] J. Kim and T. Kim, “Framework for Modeling/Simulation of Mobile Agent Systems", *Proceedings of 2000 Conference on AI, Simulation and Planning in High Autonomy Systems*, pp.53-59, 2000.

[2] J. Kim and T. Kim, “DEVS-Based Framework for Modeling/Simulation of Mobile Agent Systems”, *SIMULATION*, June 2001. (accepted for publication)

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

[4] D. B. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley. 1998.
[5] Y. Aridor and D. B. Lange, "Agent Design Patterns: Elements of Agent Application Design," *Proceedings of the Second International Conference on Autonomous Agents*, 1998.
[6] S. Franklin and A. Graesser. "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. 1996.
[7] J. Kiniry and D. Zimmerman. "A Hands-on Look at Mobile Java Agents." *IEEE Internet Computing*, July-August, 21-30. 1997.
[8] D. Wong, N. Paciorek and D. Moore. "Java-based Mobile Agents." *Communications of the ACM* 42, no. 3, 92-102. 1999.
[9] B. P. Zeigler, T. G. Kim and C. Lee. "Variable structure modelling methodology: An Adaptive computer architecture example." *Transactions of the Society for Computer Simulation*, vol. 7, no. 4, 291-319. 1991.
[10] I. Satoh, "A Formalism for Hierarchical Mobile Agents." *In Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems*, 165-172, 2000.
[11] F. J. Barros, "Modeling Formalisms for Dynamic Structure Systems," *ACM Transactions on Modeling and Computer Simulation*, vol. 7, no. 4, pp. 501-515, 1997

AUTHOR BIOGRAPHY

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), Daejeon, Korea. His research interests include methodology for modeling and simulation of discrete event systems and mobile agent systems.

TAG GON KIM received his Ph.D. in computer engineering with specialization in methodology for systems modeling/simulation from University of Arizona, Tucson, AZ, 1988. He was a Full-time Instructor at Communication Engineering Department of Bukyung National University, Pusan, Korea between 1980 and 1983, and an Assistant Professor at Electrical and Computer Engineering at University of Kansas, Lawrence, Kansas, U.S.A. from 1989 to 1991. He joined at 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/conference proceedings. He is a co-author (with B.P. Zeigler and H. Praehofer) of the book *Theory of Modeling and Simulation* (2nd ed.), Academic Press, 2000. He is the Editor-in-Chief of *Transactions of Society for Computer Simulation* published by Society for Computer Simulation International(SCS). He is a senior member of IEEE and SCS and a member of ACM and Eta Kappa Nu.