

MADESE: A Simulation Environment for Mobile Agent

Xuhui Li¹ Jiannong Cao² Yanxiang He¹ Yifeng Chen¹

¹State Key Lab of Software Engineering
Wuhan University
Wuhan, Hubei, China
{lixuhui, yxhe, yfchen}@whu.edu.cn

²Department of Computing
Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
csjcao@comp.polyu.edu.hk

Abstract

Mobile agent has attracted researchers in distributed computing for its features such as reactivity, autonomy and mobility. However, since mobile agent can react to the environment and migrate freely, it is very difficult to evaluate its performance. Further, the lack of a standard for the execution model of mobile agents also makes the performance issue ambiguous. In this paper, we build a direct execution simulation environment called MADESE for performance evaluation of mobile agents. The environment is built on a SMA model specifying mobile agent execution. Consisting of a parallel simulator with a set of built-in mobile agent systems in a LAN or one host, the environment can be used to simulate the behaviors of the mobile agents running in the Internet. We implement a prototype based on a modified Aglets platform and test the prototype with certain mobile agent-based algorithms.

1. Introduction

In the past decade, many mobile agent-based solutions have been proposed to solve problems in network computing, performance evaluation of mobile agent algorithms therefore becomes essential because it can help discover the performance and scalability bottlenecks and thus optimizes mobile agent application design. Conventional approaches, e.g. theoretical analysis, live deployment and simulation, taken to solve the performance evaluation problem in parallel and distributed algorithms, are unpractical or not efficient here due to mobile agents' features such as mobility and reactivity.

Other than conventional approaches, direct execution simulation adds realism in generic simulation. In this approach, live codes are executed in a controllable environment processing and collecting execution information. The simulation model employs real execution system so that the parallelism and

mobility embedded in algorithms under simulation can be exploited in a natural way. The approach has been proved to be effective in simulating simple mobile agent algorithms in our original prototype[1]. However, the simulation results obviously deviated from anticipation once the algorithms become complex, which was confirmed due to the variance in understanding mobile agent's semantics, especially the multi-threaded mechanism adopted by agent system. Different mobile agent systems deploy different implementation mechanisms and thus different execution models of agents. This difference leads to the ambiguity in simulation and performance evaluation of mobile agents. In fact, we have found a number of cases where the algorithm designer can not implement his algorithm correctly in a concrete mobile agent system because of the ambiguity.

To correctly design, understand and simulate a mobile agent algorithm, we must have a consistent knowledge on the execution of mobile agent behaviors. Through the previous work, we proposed a common model for describing mobile agent's execution[2]. Based on the model called SMA, we thoroughly reconstruct the simulation model and establish the new generic direct execution simulation environment called MADESE, standing for the Mobile Agent Directly Execution Simulation Environment, completely solving the semantical ambiguity problem of mobile agents and offers the users a full control over the multi-threaded mechanism in agent execution. In this paper, we would introduce the design and implementation of MADESE.

The rest of the paper is organized as follows. In Section 2, we briefly describe related works on mobile agent simulation, direct execution simulation approach, and the SMA model. In Section 3, based on SMA the simulation model and the architecture of MADESE are introduced. A prototype implementation of MADESE is described in Section 4. Finally, Section 5 concludes the paper and discusses our future works on it.

2. Related works

Direct execution simulators have been announced since early 1990s [3]. These simulators make use of available system resources to directly execute portions of the application code and simulate architectural or environmental features that are of specific interest, or are unavailable [4]. Many simulators were designed to simulate the high performance computing environment where the cost is often too high to allow a program occupy all the resource to evaluate its performance.

Some studies has been done on mobile agent simulation, literatures involving application-specific mobile agent simulation in [5, 6]. However, few works have been taken to establish the practical generic simulation model for mobile agents. J.Kim [7] once built a model to describe the mobility of agents with a Coupled-DEVS with dynamic structures, but the model is too simple and ignores the internal execution model of the agents, which make it not strong enough to simulate concrete agents. The solid simulation model must be built on the explicitly described model of mobile agent execution. Thus, we build the SMA model as the basis of building the generic simulation environment.

SMA is a model designed for specifying mobile agent execution. It adopts common concepts of mobile agent such as creation, communication, migration etc. SMA presents a simple scheme to specify the mobile agent algorithm based on asynchronous message passing. It explicitly describes the internal behaviors, especially the multi-threaded message handling mechanisms, of the agent and lays a foundation for building a simulation model for mobile agents. In [2, 8] we described the SMA model in detail and built a DEVS model for simulating mobile agents based on SMA.

3. MADESE: the simulation environment for SMA agents

3.1. Overview of the simulation model

In process simulations a process (thread) is often looked as a set of code segments separated by the interactions between processes or between the process and the environment. The discrete code segments involving no interaction and thus executing sequentially are often called local code blocks. Naturally, the simulation of a thread comprises two parts: simulation of local code blocks' execution and simulation of interactive actions. In direct execution simulation, simulating a local code block can be as simple as catching the thread's states at the end of the

block and calculating the elapsed time for the block's execution. Therefore, to simulate agent threads, we care nothing but thread's interactions which also present enough information for agent's performance evaluation.

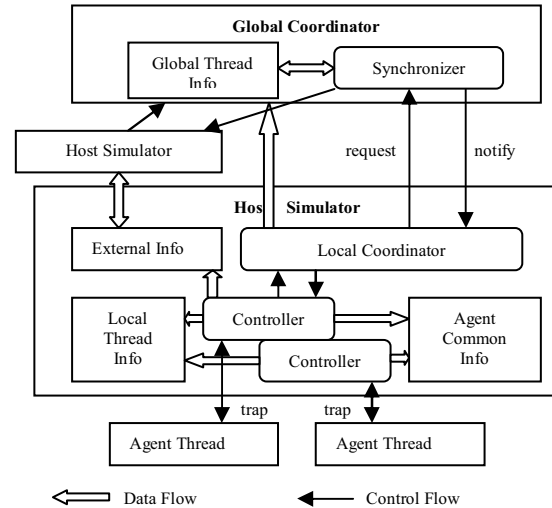


Fig. 1. Simulation model of MADESE

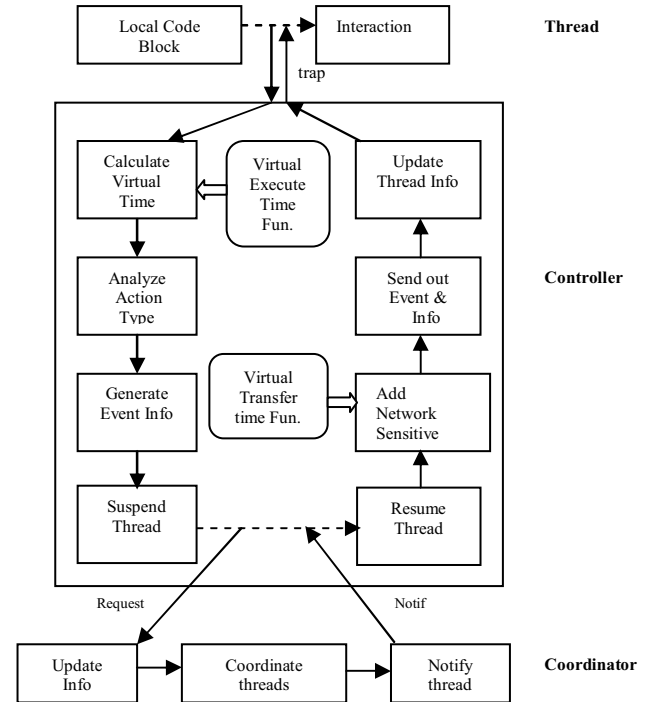


Fig. 2. Process of a trap in Controller

MADESE deploys a multi-level simulation model as depicted in Fig.1. Since the agent is logically a set of threads, we can combine the simulation of agents and hosts into a host simulator. In the model, the host simulator maintains the information of the agents and the threads, and deploys several controllers to control the threads by tracing the interactions of each thread.

With the assistance of the global coordinator and the information from other peer host simulators, the controllers schedule the threads to ensure that the whole simulation obey the law of causation. The details of controlling the threads are illustrated in Fig.2.

As Fig.2 shows, during the simulation, a working thread would be trapped into Controller when it finishes executing a local code block and prepares for an interactive action. Then the virtual time of the thread is calculated, and various logical data mainly concerning the interaction type and the associated arguments would be generated. With the virtual time and action information, the corresponding event for performance evaluation is generated, and then Controller suspends the thread, waiting for the notification from Coordinator. Once the notification arrives, the thread is resumed and the network sensitive information is generated if the action involves network communication between hosts. Before the control is returned back to the thread, the event indicating the interaction and other information would be sent out to the environment for further use. Since the controllers in a host always process similar data in the same circumstance, we use a deliberately designed controller called server controller to control all the threads in one host.

MADESE deploys a set of load simulators and a network simulator to simulate the host and the network environment. The environment simulators are simply defined as modules to provide the controllers an interface comprising a virtual time function, which calculates the duration of executing code blocks in virtual hosts or transferring data through virtual network. The environment conditions such as the load of the hosts and the contention and the traffic of the network thus depends on concrete implementations of the environment simulators.

To accurately control interactive actions and make full use of them, MADESE deploys the event mechanism and synchronization control mechanism.

3.2. Event mechanism

In MADESE, the events triggered by the interactions identify not only the boundaries of local code blocks but also the synchronization points of agents and simulation environments. Further, generally the information relating to these events plays a vital role in performance evaluation, e.g., the number and the mean length of the messages and the sites visited by an agent are often important metrics in evaluating the algorithm's performance. Sometimes more specific information such as the execution time of a code segment and the occurring time of a behavior are also concerned.

In the event model of MADESE, a set of event types are deliberately defined to support the synchronization control and record the execution status. These events include behavior and work events, environment sensitive call events, and user-defined events. An event encapsulates the event source, occurring time and some event specific data. The simulation environment generates and handles the events, and minimizes their effect on agent's execution. MADESE defines kinds of event types, including agent creation events, communication events, agent migration events, agent thread (begins, blocks, unblocks, etc.) working event, synchronization events, death events, auxiliary synchronization events, environment event and user defined events.

Here we only describe the function of environment events because the functions of the others are evident. During the simulation, the environment would provide some so-called environment sensitive interfaces for the agents to get the knowledge about the runtime environment status. The calls to these interfaces concerns the status of the simulation environment and might interfere with the execution path. Since these calls usually involve the synchronization between agents and simulation environment, MADESE defines the environment events triggered by the calls to facilitate the synchronization control.

3.3. Synchronization

The simulation environment should handle the synchronization incurred by the interactions. Two synchronization approaches, a conservative one and an optimistic one, are often used in parallel and distributed simulation systems. Since the optimistic protocols often require the environment be able to recover from the errors once the out-of-sequence event occurs, it is relatively difficult to be implemented correctly, especially in direct execution simulation. We thus adopt the conservative approach to solve the synchronization problems in MADESE.

MADESE deploys a variant of common conservative synchronization algorithm called null message algorithm in which a process would send "null" messages only comprising the synchronization information to other processes. The information, usually called "lookahead" is a time duration indicating the interval for the events to occur. Based on the lookahead, the simulator of each process can determine whether to proceed safely.

In MADESE the coordinators, a global coordinator and a set of host coordinators dispersing on every host, take charge of synchronization. Each host coordinator maintains a table recording all the agent threads in the host, and sets a timer to update the information

periodically. When the timer is time out or a thread is suspended because of an event, it updates threads' information and forward the data of the thread with the least logic time to the global coordinator. The global coordinator maintains a table of the threads with the least logic time in every host, and update the data dynamically with the information sent by the hosts. It would find the thread with the global least logic time and then send its information to all host coordinators. Once the host coordinators receive the thread information, it would calculate the lookahead of the thread and then wake up all the waiting threads in the host that can safely execute.

3.4. Architecture of MADESE

Based on the simulation model described above, we design the architecture of MADESE, as depicted in Fig.3.

MADESE introduces a master / slave architecture, consisting of two parts: *Simulation Control* and *Simulation Nodes*. *Simulation Control* configures the simulation task, acquires and analyzes the simulation result, and manages simulation nodes. Additionally, the global coordinator locates in the *Session Control* component of *Simulation Control* because of its central position.

The simulation task configuration comprises: a) environment configuration comprising the network topology in the task, the distribution of the agents' hosts and the concrete network and load simulators; b) agent tasks comprising the initial agents started in each agent host; c) performance evaluation plug-ins comprising the real time event listeners and the analyzers of the simulation results.

Simulation results are conceived by the events forwarded to *Event Manager* which dispatch them to listeners in *Monitor* to reflect the real time condition of simulation. After the simulation, the events are gathered and analyzed by the specified analyzer to evaluate agent's performance.

Session Control manage the simulation nodes by sending the control commands to them. The commands include configuring the environment in the nodes and starting or stopping the task, etc. Once receiving the commands, *Simulation Node* would start the specified agents and then transfer the control to the agent system and the components doing the simulation. A node can simulate several hosts with a set of server objects consisting of a concrete mobile agent system, a load simulator, a server controller and a host coordinator. So we can simulate many hosts in several machines in a LAN or just in a single machine.

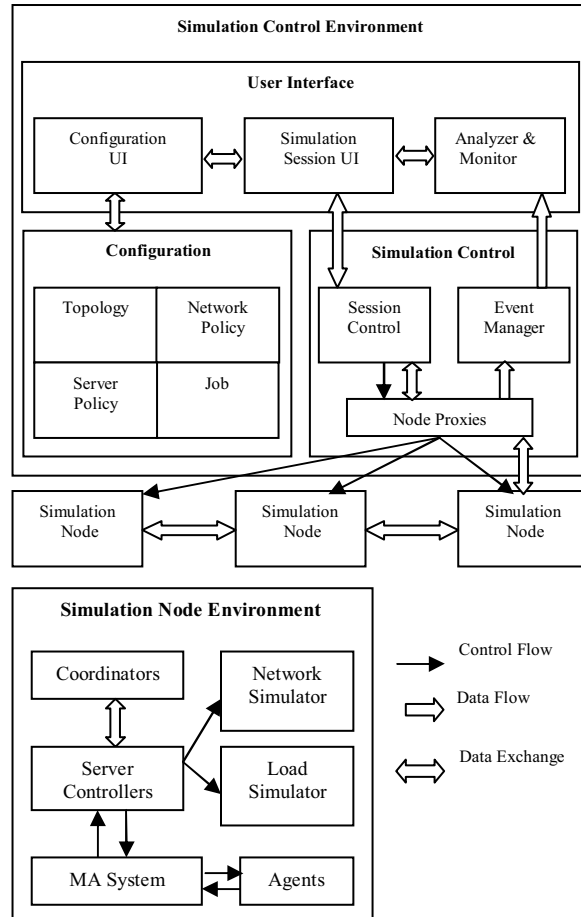


Fig.3. Architecture of MADESE

4. Implementation of MADESE

Based on the design presented above, we have built a prototype of MADESE. The prototype, called *SimulAgent*, is implemented with JDK 1.4 and the concrete mobile agent system is IBM Aglets 2.0. *SimulAgent* is composed of four parts: simulation configuration and management; simulation control and coordination; API, runtime environment and simulated environment; and monitors and analyzers. Here we briefly introduce the ideas on the runtime environment and the API, because the others are straightforward or previously have been discussed.

4.1. Runtime environment

SimulAgent rebuilt the Aglets runtime environment to make agents' execution fit for simulation control. As a separate and complete implementation of aglets runtime layer, the core of *SimulAgent* Aglets server extended the function of the standard implementation

package `com.ibm.aglets` and modified it in certain aspects. The goal of modifying the aglets package is to make the Aglet in SimulAgent work as a SMA agent and then to control its execution. Some works are done here to make the runtime layer capable of supporting the standard Aglets meanwhile adopting the SMA execution model in concrete execution.

Besides driving agents' execution, Aglets server also deploys an instance of the implementation class of Server Controller whose `trap()` method is provided for the aglets runtime layer as the entrance to the simulation environment. Thus, when an Aglet is running in SimulAgent, the execution would be trapped into the ServerController if it encounters an interaction. The processing steps of the `trap()` method, as shown in Figure 2, carry out the simulation control to agent's execution without perceptible effect.

4.2. SimulAgent API

SimulAgent provides the API package to make the aglet programs run following the SMA model. The API package mainly involves two classes: a) SAglet class, the super class for all the SimulAgent Aglets; and b) Utility class, an interface to provide some auxiliary functions.

SAglet is a direct subclass of `com.ibm.aglet.Aglet`. It extends the Aglet class with two methods, `getEnvironment()` and `getUtil()`, which return the instances of the Environment and the Utility classes. The Environment class provides a set of methods returning the environment information, e.g., the `getNetworkTopology()` method returns the simulated network topology and the `getCurrentTime()` method returns current logic time.

Utility is the most important class in the API. Many primitives in SMA are carried out by the methods provided by standard Aglet API, such as `creatagent`, `migrate`, `sendmessage` and `dispose`. The runtime environment can support the methods to follow the semantics of the corresponding primitives in SMA. However, those primitives not being supported by the Aglet API can only be implemented by SimulAgent. Thus, Utility provides some methods to fulfill the function of the primitives involving synchronization and communication such as `lock`, `unlock` and the message management primitives.

4.3. An experiment

We implemented some mobile agent algorithms and tested their performance in SimulAgent to verify the function of SimulAgent. A typical test case is a distributed mutual exclusion algorithm solved by

mobile agent which involves a certain number of migrations and message exchanges. In the algorithm, a number of nodes who want to enter the exclusive critical region dispatch their mobile agents. The agents travel among the nodes and exchange the data with them, until collecting enough information to allow its owner to enter the region safely.

We simulate the algorithm in SimulAgent. The experiment environment is set up with 5 PCs in a fast LAN, the hardware configuration of machine is PIII 900M / 256M / 20GB, the operating system is Windows 2000, and the version of Java VM is JDK1.4. We simulate the execution of the system composed of the nodes whose number ranges from 4 to 40, and make the performance evaluation based on the simulation results. The performance metrics including the average time for a node to enter the critical region, average nodes for an agent to visit, and average traffic cost for a node, under full load, middle load, and light load respectively. From the analysis of the algorithm in [9] and the following works, we know that the average cost such as the traffic and waiting time in heavy load would be smaller than it is in middle load, which is the major advantage of the algorithm. The simulation results got from SimulAgent explicitly confirmed the proposition, as shown in Fig.4.

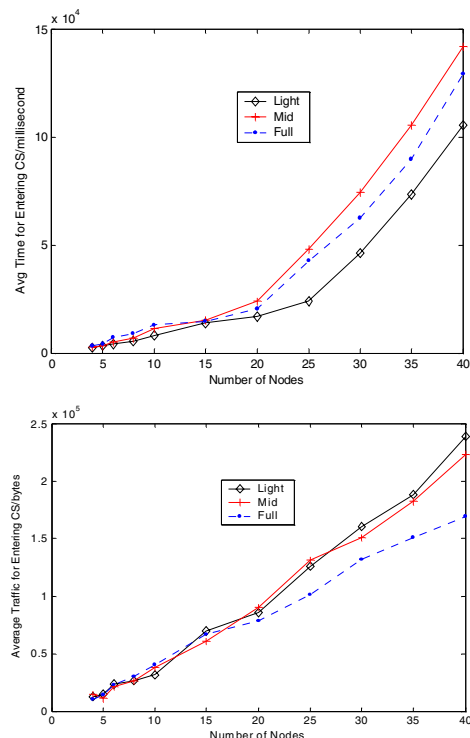


Fig.4. Simulation results of the ME algorithm

Furthermore, we also simulate the algorithm under full load with common simulation program. The

simulation program also adopts the SMA model, and the execution time of each local code block is calculated based on the same formula in SimulAgent. The comparison of the simulation results between the simulation program and SimulAgent is shown in Tab.1(M represents the result from SimulAgent, S represents the result from simulation program). The two approaches exhibit almost the same results, which provide the powerful testimony for the validity of SimulAgent.

Nodes		4	6	8	10	15
AvTime (ms)	M	3355	7218	8958	12866	14777
	S	3320	6988	8577	12740	14322
AvNodes	M	3	5.9	6.2	9.3	12.5
	S	3	5.8	6.2	9.6	12.6
AvTraffic (bytes)	M	3633	3894	4174	4398	5372
	S	3642	3886	4170	4414	5390
Nodes		20	25	30	35	40
AvTime (ms)	M	20206	42847	62483	89888	129123
	S	19695	41919	61593	89087	127694
AvNodes	M	13	14.6	16.8	18.2	18.6
	S	13.1	14.4	16.6	18.5	18.8
AvTraffic (bytes)	M	6062	6918	7875	8787	9576
	S	6077	6877	7750	8853	9662

Tab.1. Comparison of simulation results between two approaches

5. Conclusion

Performance evaluation of mobile agent algorithms remains a difficult problem because of the agent's features such as mobility, reactivity, autonomy, etc. The approaches for conventional distributed computing algorithms are not fit for the case. Further, there is a lack of a standard execution model of mobile agent, which makes its performance ambiguous. In this paper, we deployed the direct execution approach into the problem and built a direct execution simulation environment, called MADESE, for simulating mobile agents and evaluating its performance. Based on a deliberately defined execution model called SMA for mobile agents, we introduced the features and the architecture of MADESE. A prototype of MADESE is implemented in Java based on a modified version of IBM Aglets, and the experiment showed that the environment worked to good purpose.

Currently, we are engaged in the work of looking for better approaches to improve the prototype. The major aspects to improve lie in: to design efficient parallel simulation algorithms for the prototype and to make some analysis programs to calculate the

lookahead based on the program's structure; to design better host simulators and network simulators which can simulate the actual host workload and network traffic; to improve the structure of the packages in the simulagent, making it more extensible and able to accommodate different simulation algorithm, which can make comparison between the algorithms.

Acknowledgement

This research is partially supported by the Hubei Nature Science Fund in China under the contract No. 2005ABA235 and the University Grant Council of Hong Kong under the CERG Grant B-Q518.

References

- [1] X.Li, J.Cao, and Y.He, "A Direct Execution Approach to Simulating Mobile Agent Algorithms", *The Journal of Supercomputing*, 29(3), 2004. pp.171-184.
- [2] X.Li, Z.Peng, J.Cao, "A Practical Approach to Specifying and Verifying Mobile Agent Algorithms", *Journal of Pervasive Computing and Communications*, 1 (2), 2005. pp. 113-121.
- [3] J. R. Covington, S. Dwarkadas, J. Jump, S. Madala, J. Sinclair, "The Efficient Simulation of Parallel Computer Systems", *International Journal in Computer Simulation*, 1(1), 1991. pp.31-58.
- [4] S.Prakash, E.Deelman, R.Bagrodia, "Asynchronous Parallel Simulation of Parallel Programs", *Software Engineering*, 26(5), 2000. pp.385-400.
- [5] A.Lingnau, O.Drobink, "Simulating Mobile Agent Systems with Swarm", *Proc. Of First International Symposium on Agent Systems and Applications Third International Symposium on Mobile Agents*, 1999.
- [6] A. M. Uhrmacher, P. Tyschler, D. Tyschler, "Modeling and Simulation of Mobile Agents", *Future Generation Computer Systems*, 2000. pp. 107-118.
- [7] J.Kim, T.G.Kim, "DEVS-Based Framework for Modeling/Simulation of Mobile Agent Systems", *Simulation*, 76(6), 2001. pp.345-357.
- [8] X.Li, J.Cao, Y.He, J.Zhou, "A Discrete Event System Model for Simulating Mobile Agents", *Proc. Of ISPA 2005, LNCS 3758*, 2005. pp.701-712
- [9] J.Cao, X.Wang, J.Wu, "A Mobile Agent Enabled Fully Distributed Mutual Exclusion Algorithm", *Proc. 6th IEEE International Conference on Mobile Agents (MA'02)*, Spain, LNCS, 2002.