SICE Annual Conference 2010
August 18-21, 2010, The Grand Hotel, Taipei, Taiwan

A Method of Transparent Swapping Control for Mobile Agents

Masayuki HIGASHINO, Toshihiko SASAMA, Takao KAWAMURA and Kazunori SUGAHARA
Department of Information and Electronics
Graduate School of Engineering Tottori University
4-101, Koyama-Minami, Tottori 680-8552, JAPAN
Email: {s032047, sasama, kawamura, sugahara}@ike.tottori-u.ac.jp

***Abstract*—In a mobile agent system, a number of agents concentrate on one computer according to circumstances. Many mobile agent frameworks support a function of persistent agents because the computer cannot retain many agents in limited amount of memory.**

However, an agent programmer is forced to design based on careful study of the actual situation of the resources. In addition, there is a problem with inability to cope with unexpected circumstances of the resources. In this paper, we present a method of transparent persistent agents according to memory usage of an agent runtime environment. Further, we implement our proposed method, and evaluate on a practical application of mobile agent system.

***Keywords*—Mobile Agent, Swapping.**

I. INTRODUCTION

Recently computing systems have been usually composed of computers connected by a network, and that is called a distributed system. Mobile agent technology is an important issue to construct distributed systems. In a mobile agent system, a number of autonomous agents cooperate mutually and achieves given tasks. These agents are spread on some computers and migrate among these by computer networks. However, a number of agents concentrate on one computer according to circumstances. For execution of many agents on one computer at same time, a large amount of resources such as mass memory capacity and a fast CPU are required. Consequently, under the limited resource circumstances, some agents on the computer should be swapped out to second storage such as a hard disk drive.

In order to realize swapping out agents, a mobile agent framework has to support persistence of agents. Many mobile agent frameworks have been studied, and several mobile agent frameworks, such as Aglets[1] and MobileSpaces[2], support persistence of agents. However, the feature of persistence is provided as a function of called by agent. For this reason, there are the following problems:

- It requires complex programming about persistence of agents.
- An agent programmer is forced to design based on careful study of the actual situation of resources.
- An agent runtime environment crashes if it exceed memory limit unexpectedly.

In this paper, as a solution for these problems, we propose a method of transparent swapping control for a mobile agent system. This method swaps agents between memory and secondary storage automatically according to memory usage of an agent runtime environment. Thus, it is possible to develop a highly-reliable application of mobile agent system without writing program about the persistence of agents.

This paper is organized in 4 sections. The proposed design is described in Section 2. We describe the implementation in Section 3, and the evaluation on a practical application of mobile agent system in Section 4. Finally, Section 5 presents some concluding remarks.

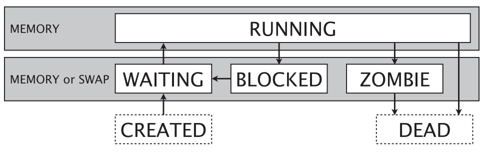

Fig. 1. Agent states.

II. DESIGN

In general, a mobile agent system consists of agents and an agent runtime environment (hereafter referred to as **ARE**). Additionally, The ARE is implemented as a process in operating system, and the agents are implemented as threads in the process. A large number of agents are performed concurrently in an ARE. Our proposal is premised on this architecture.

The remaining of this section will explain each of how to realize transparent swapping control of agents according to memory usage of an ARE.

A. Agent States

In order to realize the ARE swap-in and swap-out agents according to memory usage, the ARE needs to get a handle on each state of agent. Because some agents may be using resources which includes files and devices, and additionally, that may be in a dialogue with users of this system or the other agents. Accordingly, we defined six states of agents to determine the possibility of swapping them. Fig 1 shows an agent states.

1) CREATED: To begin with, an agent is *CREATED*. Then in order to the agent is registered in an agent scheduler, awaits acceptance of an ARE. After that the ARE assigns it the state *WAITING*. Note that in this state, the agent is not stored in memory or swap yet.

TABLE I
SWAPPING PRIORITIES.

Priority	Agent State	Description
HIGH	*ZOMBLE*	All of these agents are swapped out at one time.
MIDDLE	*BLOCKED*	These agents are swapped out in order of average amount of a blocked time.
LOW	*WAITING*	These agents are swapped out from backward of the queue.

2) WAITING: In this state, the agent is created from program codes, or cloned from the other agent, or migrated from the other ARE on the network. Note that on this occasion, if capacity of memory is filled up then, the agent is loaded into secondary storage. If not, the agent is loaded into memory. In addition, the ARE assigns an agent according to memory usage as a FIFO (first-in, first-out) method.

3) RUNNING: An agent has to be on the memory in order to execute procedures. In our proposals, the memory is managed by controlling input to the *RUNNING* from other states.

4) BLOCKED: If an agent waits a message from other agents, a thread of the agent stops, and so the agent is blocked. When the blocked agent receives the message, a thread of the agent transit to *RUNNING* through *WAITING*, and so the agent runs.

5) ZOMBIE: If an agent finishes on success or failure, a thread of the agent stops permanently. Note that on this occasion, the agent remain on a memory or a swap yet.

6) DEAD: In this state, an ARE erase the agent from a memory space and a disk space.

B. Swapping Timing

1) Swapping Out: An ARE can swap out any agents unless the agent is *RUNNING*. A swapping out is executed when an agent transition to *RUNNING*. An ARE must free some memory to execute the agent.

2) Swapping In: A swapping in is executed when an agent transition from *WAITING* to *RUNNING*.

C. Swapping Out Priority

In order to improve execution efficiency, our proposal method defines swapping priorities as shown TABLE I. An ARE swaps out agents in order of this priorities.

III. IMPLEMENTATION

We have implemented our proposal method on Maglog[3], [4]. Maglog is our proposal mobile agent framework, and we took up as an example of a mobile agent framework. It is based on Prolog and is implemented by extending PrologCafé[5], which is a Prolog-to-Java source-to-source translator system. The ARE is implemented in Java and runs on any platform providing a Java Runtime Environment(JRE). The agents are java objects which can run concurrently using threads. Using Java Object serialization, the ARE migrate and swap-in/swap-out agents. Fig 2 shows an implementation of the agent scheduler that based on our proposal architecture.

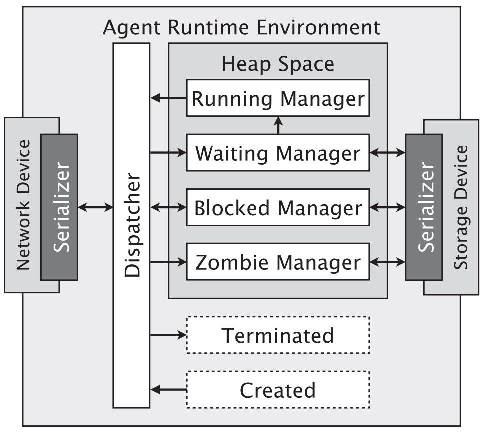

Fig. 2. An implementation of an agent scheduler that based on our proposal architecture.

IV. EXPERIMENTS

In order to confirm an effect of our proposal method, we applied that to our distributed e-Learning system[6] as an example of a practical application. The system is based on P2P architecture and every user's computer plays the role of a client and a server. In addition, each exercise is implemented as an agent. However, in this system, it often happens that many agents run at the same time on the same computer. Then the computer kept crashing as a P2P node. In experiments, we set a capacity of the memory usage to 64MB in spite of the total size of agents is 300MB. Then, we concentrate all agents on the one computer. As a result, the system has run stably without an out of memory crash.

V. CONCLUSION

In this paper, we have proposed and implemented a method of transparent swapping control on a mobile agent framework. In order to confirm the effect of our proposal method, we have applied that to a practical distributed system with a mobile agent technology. As a result, all nodes in the system have run without an out of memory crash.

REFERENCES

[1] D. B. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley, 1998.
[2] I. Satoh, "Mobilespaces: A framework for building adaptive distributed applications using a hierarchical mobile agent system," in *Proc. IEEE International Conference on Distributed Computing Systems*. IEEE Press, April 2000, pp. 161–168.
[3] S. Motomura, T. Kawamura, and K. Sugahara, "Logic-based mobile agent framework with a concept of "field"," *IPSJ Journal*, vol. 47, no. 4, pp. 1230–1238, 4 2006.
[4] T. Kawamura, S. Motomura, and K. Sugahara, "Implementation of a logic-based multi agent framework on java environment," in *Proceedings of IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, H. Hexmoor, Ed., 4 2005, pp. 486–491, waltham, Massachusetts, USA.
[5] M. Banbara and N. Tamura, "Translating a linear logic programming language into Java," in *Proc. ICLP'99 Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages*, M.Carro, I.Dutra *et al.*, Eds., December 1999, pp. 19–39.
[6] T. Kawamura and K. Sugahara, "A mobile agent-based p2p e-learning system," *IPSJ Journal*, vol. 46, no. 1, pp. 222–225, 1 2005.