Realization of Persistency in a Multi-Agent Framework

Shinichi MOTOMURA and Junya KISHIDA
The Graduate School of Engineering, Tottori University
4–101, Koyama-Minami, Tottori 680–8552, JAPAN
motomura@tottori-u.ac.jp

Takao KAWAMURA and Kazunori SUGAHARA
Department of Information and Knowledge Engineering
Tottori University
4–101, Koyama-Minami, Tottori 680–8552, JAPAN
+81 857 31 5217
{kawamura,sugahara}@ike.tottori-u.ac.jp

Abstract— *In this paper, we present persistency of an agent runtime environment and generated agents. Maglog which is implemented in Java environments is taken up as an example of the agent runtime environment. Maglog consists of three basic components, which are agents, agent servers and fields. Agent server and fields are corresponding to agent runtime environments. The persistency is used for the following two purposes. One is for suspending agents. The other is for suspending a system which is developed by using Maglog.*
For example, when a computer will be stopped while a system is running, agents are suspended then their suspended state is stored. Subsequently, the agent server is suspended then the suspended state and fields are stored. When the system is restarted, first, the agent server resumes then the agents resume.

1.INTRODUCTION

Multi-agent system is drawing attention as a structural model for many software systems including distributed systems and artificial intelligence systems[1]. In general, a multi-agent system consists of agents and an agent runtime environment (hereafter referred to as ARE). An ARE provides required functions for agents. In a multi-agent system, a number of autonomous agents cooperates 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 the circumstances. For execution of these agents on one computer at same time, a large amount of resource such as mass memory capacity and a fast CPU is 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 agent swapping out functions to hard disks, multi-agent frameworks should support persistency of agents. Several multi-agent frameworks, such as Aglets[2] and MobileSpaces[3], support persistency of agents. However, in these multi-agent frameworks, when an agent executes a program which describes the behavior of the agent, the agent can suspend but can not resume from the stopped point.

The persistency of agents that we require is described as follows.

Suspend When an agent suspends, the agent and its suspended state are written in hard disks.

Resume When the agent resumes,according to the information written in the hard disk. The agent restarts from the stopped point.

Besides, if a system which is developed by a multi-agent framework tries to suspend, not only agents but also an ARE has to support persistency. However, there are no multi-agent frameworks which support persistency of an ARE.

In multi-agent systems, mobility of agents is important because of not only reducing network latency but simplifying architecture of software systems[4]. Therefore, the following situations must be considered.

Situation 1 An agent suspends on a computer where the agent was created.

Situation 2 An agent suspends on a computer where the agent was not created.

Situation 3 When an agent tries to come back to a computer, the computer has stopped.

First situation is solved easily. In this paper, the second situation is solved. In the third situation, when an agent cannot come back to a computer, the agent has to know the reason why the system was suspended or was broken down. In order to solve the problem, when an agent cannot come back to a computer, the agent should sleeps. Then, when the computer restarts, the agent should be woken up. Multi-agent frameworks should support the function, which agents are woken up when suspended computer restarts. However, the function is not implemented yet.

We examine that persistency of agents and an ARE is introduced into a multi-agent framework which is implemented in a Java environment. Maglog[5] which is our proposed multi-agent framework is took up as an example of a multi-agent framework. It is based on Prolog and is implemented by extending PrologCafé[6], which is a Prolog-to-Java source-to-source translator system.

This paper is organized in 5 sections. The design of persistency in a multi-agent framework is explained in Section 2. We describe the implementation of in Section 3 and experimental results in Section 4. Finally, some concluding remarks are drawn in Section 5.

2.DESIGN

A multi-agent framework consists of agents and an ARE in general. Agents and an ARE consist of Java objects in multi-agent frameworks, which are implemented in a Java environment. An agent is an autonomous program and a thread on an ARE. The characteristic of an ARE is to provide the following functions for agents:

1. Management agents, such as creation and destruction,
2. Communication with other agents,
3. Migration to another computer in a network,
4. Accessing an operation system's functions, such as reading/writing files.

The remaining of this section will explain each of how to realize persistency of agents and how to realize persistency of an ARE.

Persistency of Agent

1. Suspending/Resuming Timing

   Figure 1 shows an agent's life cycle while an agent is destroyed after the agent is created. To begin with, an agent is created then the agent is activated. Next, in order to execute procedures which the agent has, the agent runs. If the 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 runs again, and so the agent runs. Agents run autonomously, so that agents can suspend when only agents are running. There are the following situations when agents try to suspend.

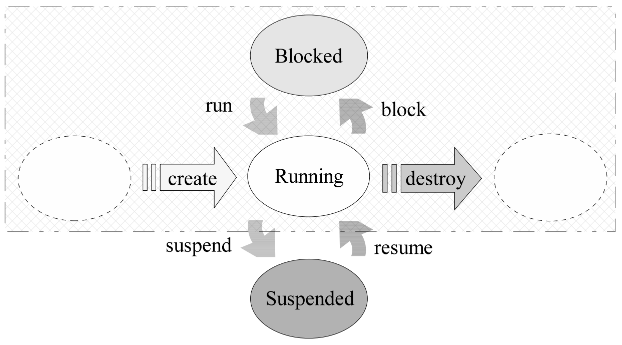


Figure 1 - Agent's life cycle.

   (a) An agent executes a procedure to suspend the agent.
   (b) An agent receives an offer of which other agents request to suspend the agent.
   (c) An agent receives an order of which an ARE requests to suspend the agent.

   If an agent receives a suspending order from an ARE when the agent is blocked, the agent runs again then suspends. After the agent resumes, the agent is blocked again.

   On the other hand, an agent is resumed by an ARE in the following situations.

   (a) A user or another agent requests to resume the agent.
   (b) An ARE decides to resume the agent. For example, if an ARE suspends agents which are under low load, the ARE will resume the agents later.

2. Required Mechanisms

   Agents consist of Java objects. In order to realize persistency of agents, Java objects are serialized. The general contract for serializing Java objects is specified in the Java Object Serialization Specification[7]. Object Serialization is a mechanism built into the core Java libraries for writing a graph of objects into a stream of data. However, Object Serialization is not enough. If no custom Object Serialization is used, objects of an agent may not be deserialized on a host where the agent was not created. Because the class descriptions of the objects may not be in the host. Therefore, Object Serialization is customized with dynamic class loaders. A dynamic class loader contains bytecodes of Java classes of an agent and loads the Java classes if necessary. Details of the custom Object Serialization and the dynamic class loader are described in another paper[8].

3. Steps of Suspending/Resuming

   Figure 2 shows an overview of persistency mechanisms. When an agent executes a procedure to suspend, the

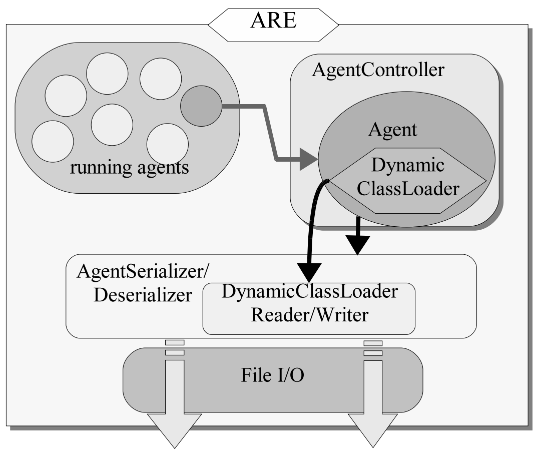


Figure 2 - An overview of agent's persistency mechanisms.

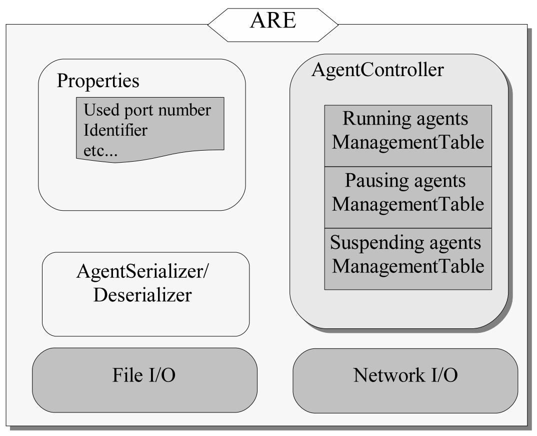

ARE
Properties
Used port number
Identifier
etc...
AgentController
Running agents ManagementTable
Pausing agents ManagementTable
Suspending agents ManagementTable
AgentSerializer/ Deserializer
File I/O
Network I/O

Figure 3 - An overview of an ARE.

agent calls a function in the ARE on which it is running, the following steps are performed.

(a) AgentController receives the call then dispatches the agent to AgentSerializer, which implements the custom Object Serialization.

(b) AgentSerializer retrieves the dynamic class loader object from the agent. The dynamic class loader object is serialized then is written in a file.

(c) AgentSerializer serializes objects of the agent then writes in a file.

(d) AgentController stops a thread of the agent. In order to be able to resume, AgentController knows the suspended agent's information, such as an agent's identifier and about above files.

When an ARE resumes an agent, the following steps are performed.

(a) AgentController gets the dynamic class loader object which is deserialized by AgentDeserializer.

(b) AgentController gets objects of the agent which are deserialized by AgentDeserializer with the dynamic class loader object.

(c) AgentController runs a thread of the agent.

Persistency of ARE

An ARE is suspended when a user or an agent requests to suspend. Although an ARE also consists of Java objects, a whole of it can not be serialized so that some components can not be serialized. Figure 3 shows an overview of an ARE. In this figure, components to provide network communication and accessing files cannot be serialized. Properties and AgentController of the other components have to be serialized, AgentSerializer and AgentDeserializer are not necessary to be serialized. When an ARE is suspended, the following steps are performed.

1. AgentController gives a suspending order to all agents then waits for until the all agents complete suspending.
2. The ARE serializes serializable components then writes them in files.
3. The ARE shuts down.

When a user wants to run an ARE again, the ARE is resumed. The resuming are performed as follows.

1. A user starts up the ARE with a parameter to resume.
2. The ARE reads the files then deserializes the components.
3. The ARE resumes, then AgentController resumes all suspended agents.

3.IMPLEMENTATION

To confirm the behaviors, we implement persistency of agents and AREs in our multi-agent framework Maglog.

Overview of Maglog

Maglog is a multi-agent framework based on Prolog and is implemented in a Java environment. Figure 4 shows an overview of a multi-agent system built by using Maglog. Maglog consists of three basic components, which are agents, agent servers and fields. An agent server and fields provide functions of as an ARE.

1. Agent

   An agent has the following functions:

   (a) Execution of a program that describes the behavior of the agent,

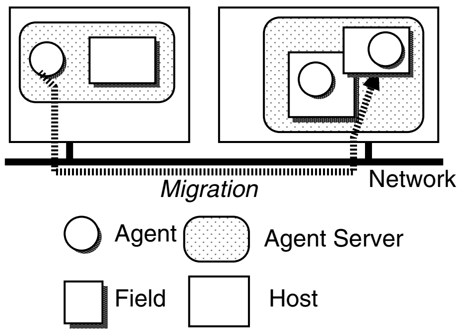


Figure 4 - Overview of a multi-agent system built using Maglog.

(b) Execution of procedures stored in a field where the agent is currently located,

(c) Communication with other agents through a field,

(d) Creation of agents and fields,

(e) Migration to another host in a network.

An agent has Prolog interpreter because the agent resumes from its stopped point. If an agent is running on JVM directly without Prolog interpreter, when the agent can not resume from the stopped point. Because Java API does not provide methods to access program counter and stack in order to access a thread's execution state. In contrast, Maglog has the WAM[9] as Prolog interpreter which is an abstract machine tailored to Prolog.

2. Agent Server

   An agent server is a runtime environment that provides required functions for agents. For example, an agent server provides a migration function. When an agent migrates from host-A to host-B, the agent server on host-A suspends the agent's execution and transports the agent to host-B. After that, the agent server on host-B resumes execution of the agent. An agent server also manages fields and provides functions that enable an agent to utilize them.

3. Field

   A field is an object managed by an agent server to hold Prolog clauses. An agent communicates with other agents indirectly through fields. An agent can communicate with other agents not only asynchronously but also synchronously.

Persistency of Agent

Figure 5 shows a UML diagram which is an overview of classes of Maglog. ClassLoader and CodeLoader are corresponded to DynamicClassLoader. AgentScheduler is corresponded to AgentController. AgentServerRemote provides network functions which are using Java RMI and XML-RPC.

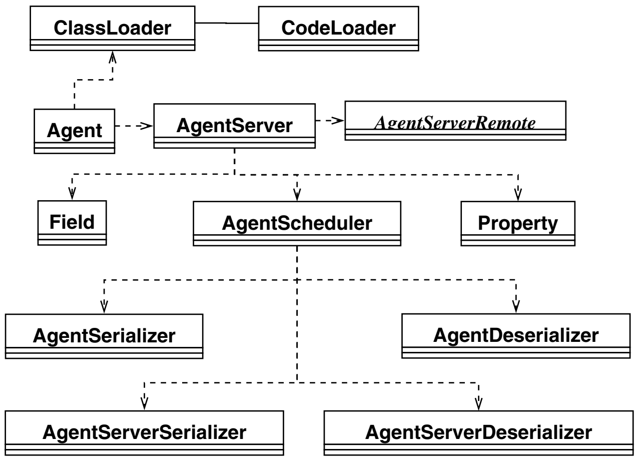
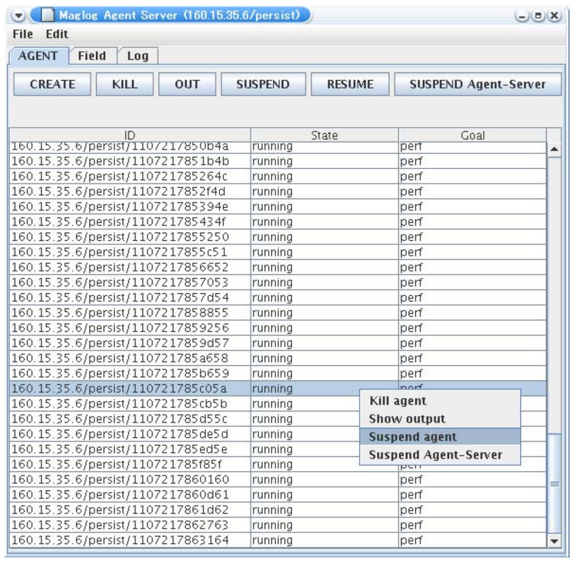
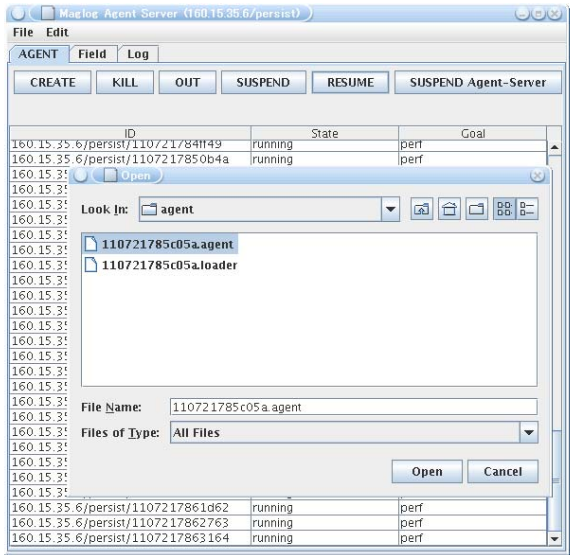


Figure 5 - A UML diagram which is an overview of classes of Maglog



ID	State	Goal
160.15.35.6/persist/1107217850b4a	running	perf
160.15.35.6/persist/1107217851b4b	running	perf
160.15.35.6/persist/110721785264c	running	perf
160.15.35.6/persist/1107217852f4d	running	perf
160.15.35.6/persist/110721785394e	running	perf
160.15.35.6/persist/110721785434f	running	perf
160.15.35.6/persist/1107217855250	running	perf
160.15.35.6/persist/1107217855c51	running	perf
160.15.35.6/persist/1107217856652	running	perf
160.15.35.6/persist/1107217857053	running	perf
160.15.35.6/persist/1107217857d54	running	perf
160.15.35.6/persist/1107217858855	running	perf
160.15.35.6/persist/1107217859256	running	perf
160.15.35.6/persist/1107217859d57	running	perf
160.15.35.6/persist/110721785a658	running	perf
160.15.35.6/persist/110721785b659	running	perf
160.15.35.6/persist/110721785c05a	running	perf
160.15.35.6/persist/110721785cb5b	running	
160.15.35.6/persist/110721785d55c	running	
160.15.35.6/persist/110721785de5d	running	
160.15.35.6/persist/110721785ed5e	running	
160.15.35.6/persist/110721785f85f	running	perf
160.15.35.6/persist/1107217860160	running	perf
160.15.35.6/persist/1107217860d61	running	perf
160.15.35.6/persist/1107217861d62	running	perf
160.15.35.6/persist/1107217862763	running	perf
160.15.35.6/persist/1107217863164	running	perf

Figure 6 - A screen-shot of the user interface program when a user tries to suspend an agent.



ID	State	Goal
160.15.35.6/persist/110721784ff49	running	perf
160.15.35.6/persist/1107217850b4a	running	perf
160.15.35.6/persist/1107217861d62	running	perf
160.15.35.6/persist/1107217862763	running	perf
160.15.35.6/persist/1107217863164	running	perf

File Name: 110721785c05a.agent
Files of Type: All Files

Figure 7 - A screen-shot of the user interface program when a user tries to resume an agent.

Table 1 - The experimental conditions.

CPU	Intel Xeon 3GHz
Memory	1GB
OS	TurboLinux 10 Desktop
JRE	1.4.2

An agent can suspend by executing the predicate, which name is *suspend*. When an agent executes *suspend* predicate, objects of the agent are dispatched to AgentSerializer via AgentServer. AgentSerializer writes serialized ClassLoader object and serialized CodeLoader objects in a file, which name is generated using the agent's identifier. The file is called SuspendedClassLoaderFile. Then AgentSerializer writes serialized Agent object in a file, which name is generated using the agent's identifier. The file is called SuspendedAgentFile. Finally, AgentScheduler stops the thread of the agent.

A user can suspend an agent and can resume the agent by a user interface program for manipulation of agent servers. Figure 6 shows a screen-shot of the user interface program when a user tries to suspend an agent. If the thread of the agent waits or sleeps, the thread is woken up by AgentScheduler using the Java notify method, then objects of the agent are serialized.

Figure 7 shows a screen-shot of the user interface program when a user tries to resume an agent. The user chooses the SuspendedAgentFile. AgentDeserializer reads the file then reads the SuspendedClassLoaderFile. AgentDeserializer deserializes objects of the agent using above read files then dispatches deserialized objects to AgentScheduler. The thread of the agent is started by AgentScheduler.

Persistency of Agent Server and Field

A user can suspend an agent server by the user interface program and can resume the agent server with a command-line option, which is `--resume`. When a user tries to suspend an agent server, the user clicks the *SUSPEND Agent-Server* button in Fig. 6. First, AgentScheduler serializes objects of all agents by AgentSerializer. Secondly, objects of Field and objects of Property are serialized by AgentServerSerializer then are written in a file. The file is called SuspendedAgentServerFile. Finally, the agent server shuts down. When a user starts the agent server again with the command-line option, AgentServerDeserializer reads the file then deserializes the objects of Field and objects of Property. AgentScheduler deserializes objects of all serialized agents by AgentDeserializer, then threads of the agents are started.

4. EXPERIMENTS

This section presents experimental results. Table 1 shows the experimental conditions. In one experiment, we examine the elapsed time and size of the generated files when an agent is suspended. Subsequently, we examine the elapsed time when the agent resumes. The agent's class file size is 1.2KB. Table 2 shows the elapsed times of suspending and resuming. Table 3 shows SuspendedAgentFile size and SuspendedClassLoaderFile size.

In the other experiment, we examine the elapsed time of suspending and SuspendedAgentServerFile size when an agent server is suspended. The 100 agents are running on the agent server. Subsequently, we examine the elapsed time when the agent server resumes. The agent's class file size is 1.2KB. Table 4 shows the elapsed times of suspending and resuming. The SuspendedAgentServerFile size is 69KB.

Table 2 - The elapsed time of suspending an agent and resuming the agent.

Suspending time	Resuming time
280 msec	154 msec

The agent's class file size 1.2KB.

Table 3 - The SuspendedAgentFile size and the SuspendedClassLoaderFile size.

SuspendedAgentFile	SuspendedClassLoaderFile
66.8 KB	2.2 KB

The agent's class file size 1.2KB.

Table 4 - The elapsed time of suspending an agent server and resuming the agent server.

Suspending time	Resuming time
36,175 msec	10,426 msec

Each agent's class file size 1.2KB.

5.Conclusion

We have implemented persistency of agents, agent servers and fields in Maglog for the following two purposes. One is for suspending agents. The other is for suspending a system which is developed by using Maglog. For example, when a computer will be stopped while a system is running, agents are suspended then their suspended state is stored. Subsequently, the agent server is suspended then the suspended state and fields are stored. When the system restarts, first, the agent server resumes then the agents resume.

In order to realize persistency of agents, Maglog supports customized Object Serialization and the dynamic class loading mechanism. Agent servers do not consist of only serialized components, and so agent server's persistency steps which serialized components are serialized and an agent server shuts down are defined.

In Maglog, it often happens that many agents run at the same time. Using the persistence of agents, it will be added in the future that agents which are under low load are swapped out.

References

[1] Weiss, G.(ed.): *Multi-Agent Systems: A Modern Approach to Artificial Intelligence*, MIT Press (2000).

[2] Lange, D. B. and Oshima, M.: *Programming and Deploying Java Mobile Agents with Aglets*, Addison Wesley (1998).

[3] Satoh, I.: MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System, *Proceedings of the IEEE International Conference on Distributed Computing Systems*, IEEE Press, pp. 161–168 (2000).

[4] Lange, D. B. and Oshima, M.: Seven good reasons for mobile agents, *Communications of the ACM*, Vol. 42, No. 3, pp. 88–89 (1999).

[5] Motomura, S., Kawamura, T. and Sugahara, K.: Logic-Based Mobile Agent Framework with a Concept of "Field", *IPSJ Journal*, Vol. 47, No. 4, pp. 1230–1238 (2006).

[6] Banbara, M. and Tamura, N.: Translating a Linear Logic Programming Language into Java, *Proceedings of the ICLP' 99 Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages* (M.Carro, I.Dutra et al.(eds.)), pp. 19–39 (1999).

[7] Sun Microsystems: The Java^{TM} Object Serialization Specification, Web page. http://java.sun.com/j2se/1.5.0/docs/guide/serialization/.

[8] Motomura, S., Kawamura, T. and Sugahara, K.: Combination of XML-RPC and Mobile Agent Technologies, *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pp. 552–557 (2006). Dallas, Texas, USA.

[9] Aït-Kaci, H.: Warren's Abstract Machine A Tutorial Reconstruction (1999).