Proceedings of the 17th IASTED International Conference
Parallel and Distributed Computing and Systems
November 14-16, 2005, Phoenix, AZ, USA

MAGLOG : A MOBILE AGENT FRAMEWORK FOR DISTRIBUTED MODELS

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

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

ABSTRACT

A novel distributed model characterized by the following features is proposed in this paper. Each computer has client functions concurrently with server functions assigned. Every computer is able to join and to leave the system flexibly. While server functions of the system do not change as a whole, they are shared on each computer flexibly. That is, the server functions of each computer vary according to the conditions of computers joined to the system.

To implement our model, the new framework named Maglog for mobile agent systems is designed and developed on Java environment. In Maglog, a concept called "field" is introduced. By using this concept, the following functions are realized. Firstly, the agent migration which is the function that enables agents to migrate between computers. Secondly, the inter-agent communication which is the indirect communication with other agents through the field. That is, an agent is able to import data or programs that other agents stored into the field. Finally, the adaptation which is the function that enables agents to execute programs stored in the field.

The effectiveness of the proposed model and Maglog are confirmed through the demonstrations of two applications: the distributed e-Learning system and the schedule arrangement system.

KEY WORDS

Distributed Software Systems and Applications, Mobile Agent, P2P, Java

1 Introduction

In the construction of network application systems, the client/server model is widely adopted. Client computers are able to utilize the services fulfilled by a server in this model. It is easy to construct systems based on the client/server model, however, the following problems should be concerned carefully. Firstly, if the number of clients increases, the load of the server will be heavy and a response time will go down. Secondly, if a server crushes, all services will be stopped. To solve these problems, systems consist of several servers which manage identical services have been developed. However, this solution is effective within a certain number of clients. Similar problems will occur again when the number of client computers increases more. In this paper, a novel distributed model based on mobile agent technologies is proposed. The proposed model is characterized by the following features.

1. Each computer has client functions concurrently with server functions assigned.
2. Every computer is able to join and to leave the system flexibly.
3. While server functions of the system do not change as a whole, they are shared on each computer flexibly. That is, the server functions on each computer vary according to the conditions of computers joined to the system.

In ordinal P2P models, the functions above are equipped except the last one, which is tried to be equipped in our model.

In order to implement our model, mobile agent technologies are suitable. In our model, agents are able to have client and server functions simultaneously, and to migrate between computers with data and programs. For realization of the mobile agent systems, the following functions are required to be implemented.

1. Agents should be able to migrate from one computer to another with data and programs.
2. Agents should be able to communicate with other agents.
3. Agents should be able to adapt themselves to environments such as computers they belong to. The adaptation is accomplished by taking data and programs of the environments into themselves.

Considering the above points, a distributed model based on agent technologies and a concept of the "field" are proposed. The field is utilized to realize the required functions simply.

Agents communicate with other agents indirectly through the field and adapt themselves to the environment by importing data and programs stored in the field. The functions realized by the field are summarized as follows.

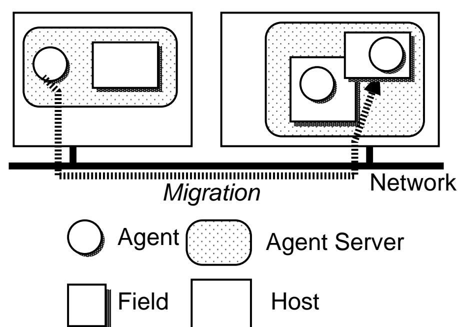

Figure 1. Overview of a mobile agent system on Maglog.

1. Migration: Function that enables agents to migrate between computers.
2. Inter-agent communication: Indirect communication with other agents through the field. That is, an agent is able to import data or programs that other agents stored in the field.
3. Adaptation: Function that enables agents to execute programs stored in the field.

For the implementation of the mobile agent system with the concept of the field, programs that describe behavior of the agent are written in Prolog language in our system. Since Prolog is logic programming language and has powerful pattern matching mechanism, agents are able to search data and programs stored in fields easily. This powerful pattern matching mechanism of Prolog is called unification. Unifications between computers are realized to construct mobile agent system.

Although several mobile agent frameworks such as Aglets[1], Mobilespaces[2], Jinni[3] and MiLog[4] have been proposed, these frameworks do not have the concept of the field. In order to realize our model, these frameworks are insufficient because agents do not have functions to adapt themselves to environments. Flage[5] has similar concepts, however, they do not realize the above-mentioned functions. Besides, it cannot realize backtracking and unification accomplished across a network.

In this paper, the mobile agent framework named Maglog on Java environment is proposed to implement the above-mentioned features. Java is adapted because of its huge class libraries to build network applications. It also should be noted that Java's goal of "write once, run anywhere" is desirable for mobile agent systems.

2 Overview of Maglog

Figure 1 shows an overview of a mobile agent system on Maglog executing an example. In the figure, two computers (hereafter referred as hosts) are connected to a network and agent servers are running on each of them to activate agents and to provide fields for them.

The rest of this section describes three basic components of Maglog, that is, agent, agent server and field.

2.1 Agent

An agent has the following functions.

1. Execution of a program that describes behavior of the agent
2. Execution of procedures stored in a field where the agent currently locates
3. Communication with other agents through a field
4. Creation of agents and fields
5. Migration to another host in a network. The model of agent state migration in Maglog is strong migration which involves the transparent migration of agent's execution state as well as its program and data.

For creation of a child agent, a parent agent executes the following built-in predicate.

```
create(AgentID,File,Goal)
```

In this predicate, `File` corresponds to the filename in which the behavior of the agent is described. If the execution of the predicate is successful, an agent is created and its globally unique identifier `AgentID` is returned. The created agent immediately executes the goal specified by the argument `Goal` and disappears when the execution is accomplished.

An agent can obtain its identifier by executing the following built-in predicate.

```
get_id(Agent)
```

Each agent contains Prolog program and its interpreter. Initial behavior of the agent is described in the Prolog program given by `File` in the predicate of its creation. Since Prolog language treats programs and data identically, the agent behavior might be modified during execution. For example, as shown in Fig. 2, agentA obtains a clause '`p(X):-q(X),r(X).`' from fieldA. That is to say, an agent is able to import clauses from fields so that it can change its behavior dynamically.

The built-in predicate `in/2` will be described in Section 3.1. Here the notation `Name/Arity` is the predicate indicator (hereafter referred as PredSpec) which refers one or several predicates. `Name` and `Arity` correspond to the name of predicate and its number of argument respectively.

2.2 AgentServer

An agent server is a runtime environment for agents and it provides required functions for agents. The above-mentioned predicates, such as `create/3` and `get_id/1`, are the examples of the functions.

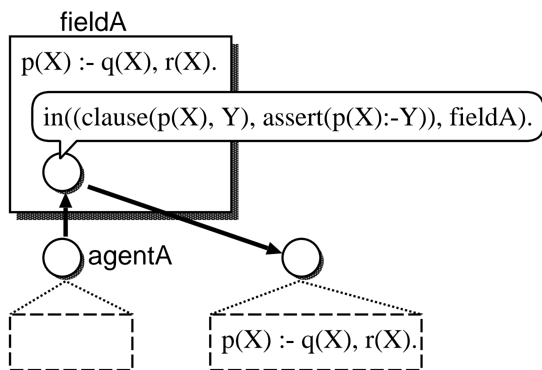

Figure 2. Dynamic change of program that describes behavior of the agent by asserting a new clause.

An agent server creates and deletes agents. An agent server assigns `AgentID` to the created agent. `AgentID` consists of host's IP address and the time the agent created, so that it becomes globally unique. In addition, an agent server also provides an agent migration function. When an agent migrates from hostA to hostB, the agent server on hostA suspends the agent's execution and transports the agent to hostB. And after that the agent server on hostB resumes the execution of the agent.

An agent server also manages fields and provides functions for an agent to utilize them.

2.3 Field

A field is an object managed by an agent server to hold Prolog clauses, and it is created when an agent executes the following built-in predicate.

```
fcreate(Field)
```

If `Field` is an unbound variable, a field which has a unique identifier is created, and its identifier is bound to the argument `Field`. If `Field` is a symbol, the action of this predicate depends on whether the field whose identifier is the symbol exists or not. If it does not exist, a field whose identifier is the symbol is created, otherwise nothing is done.

Important features of Maglog realized with the concept of the field will be described in the following section.

3 Features realized with Field

3.1 Predicate Library

An agent enters a field and executes a goal by the following built-in predicate.

```
in(Goal, Field)
```

The agent exits from `Field` automatically whether the execution succeeds or not. This built-in predicate is re-executable, i.e. each time it is executed, it attempts to enter the field and executes the next clause that matches with `Goal`. When there is no more clause to execute, this predicate fails.

When an agent enters into a field, it imports procedures of the field and combines them with procedures of itself. Therefore, an agent needs not hold all of the program by itself to solve a problem, but rather enters the appropriate fields which provide necessary procedures. An agent can change its behavior dynamically according to the field which it entered. In this way, an agent can adapt its behavior to the environments.

Figure 3 shows an example that an agent executes different `print/1` predicates in fieldA and fieldB. The execution of the goal `print('Hello!')` sends the string "Hello!" to a printer when the agent is in fieldA, on the other hand, the same goal creates a new window containing the string "Hello!" when the agent is in fieldB. Because fieldA and fieldB provide an appropriate procedure for their output devices.

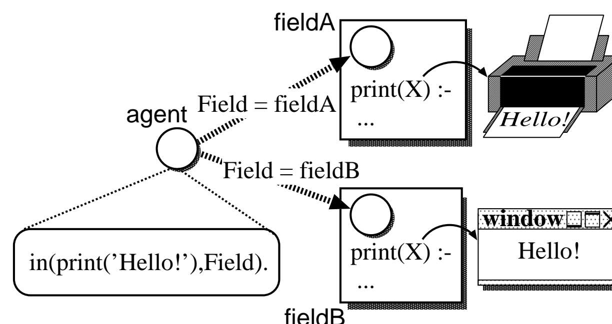

Figure 3. Dynamic Change of agent's behavior according to the field.

3.2 Inter-agent Communication

Agents entering the same field can be considered of forming a group. The procedures within the field are shared by the agents. Moreover, by adding/removing procedures within the field, agents can influence the behavior of other agents.

Updating procedures in a field can be done by the following built-in predicates.

```
fasserta(Clause, Field)
fassertz(Clause, Field)
fretract(Clause, Field)
```

The first argument `Clause` of these predicates is a clause to be added or removed from the field specified by the second argument `Field`. `fasserta/2` inserts the clause in

front of all the other clauses with the same functor and arity. Functor and arity mean the name of predicate and its number of the argument respectively. On the other hand, `fassertz/2` adds the clause after all the other clauses with the same functor and arity. `fretract/2` removes the next unifiable clause that matches with the argument from the field. This built-in predicate is re-executable, i.e. each time it is executed it attempts to remove the next clause that matches with its argument. If there is no more clause to remove, then this predicate fails.

By using these predicates, an agent can communicate with other agents not only asynchronously but also synchronously. An agent has two modes for execution of procedures stored in a field. In the fail mode, the execution fails when an agent attempts to execute or to retract a non-existent clause in a field. In the block mode, an agent that attempts to execute or to retract a non-existent clause in a field is blocked until another agent adds the target clause to the field. For agents in the block mode, a field can be used as a synchronous communication mechanism such as a tuple space in Linda model[6]

Figure 4 shows an example of the synchronous inter-agent communication.

1. `PARENT` creates `fieldA`.
2. `PARENT` creates `CHILD` and makes it execute `main('fieldA')`. `PARENT` attempts to remove the clause that matches `ans(ID,X)` from `fieldA` and it is blocked until a unifiable clause is added by `CHILD`.
3. `CHILD` executes `calculate(X)` and the result is bound to `X`. The identifier of `CHILD` is bound to `ID` by the execution of the built-in predicate `get_id(ID)`. `CHILD` adds `ans(ID,X)` to `fieldA`.
4. `PARENT` wakes up and removes `ans(ID,X)` from `fieldA`.

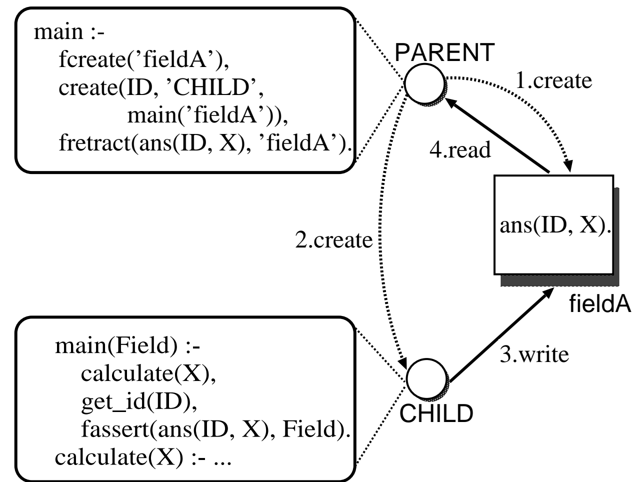

Figure 4. Agents can communicate synchronously through a field.

3.3 Agent Migration

Each agent server has globally unique identifier that is composed of the server IP address and defined name.

If the second argument of the predicates `in/2`, `fasserta/2`, `fassertz/2`, and `fretract/2` is specified in the form of `Field@ServerID`, the agent executing this predicate migrates to the host in which the agent server specified by `ServerID` runs, and enters `Field`. The agent returns to the host located before the migration automatically as it exits the field.

Figure 5 shows that the agent matches `f(X)` with clauses in two fields in hostA and hostB. As shown in Fig. 5, this attempt proceeds through performing the following steps and succeeds.

1. An agent enters fieldA in hostA and executes the goal `f(X)`. Consequently `X` is bound to 3, because `f(3)` is the first clause that matches with `f(X)`.
2. The agent migrates to hostB and enters fieldB.
3. The agent executes the goal `f(3)`. This attempt fails since there is no clause that matches with `f(3)`.
4. The agent returns to hostA and enters fieldA automatically.
5. The agent attempts to execute the next clause that matches with `f(X)`. `X` is therefore bound to 5.
6. The agent migrates to hostB and enters fieldB, again.
7. The agent executes the goal `f(5)`. This attempt succeeds this time since there is the clause `f(5)` in fieldB.

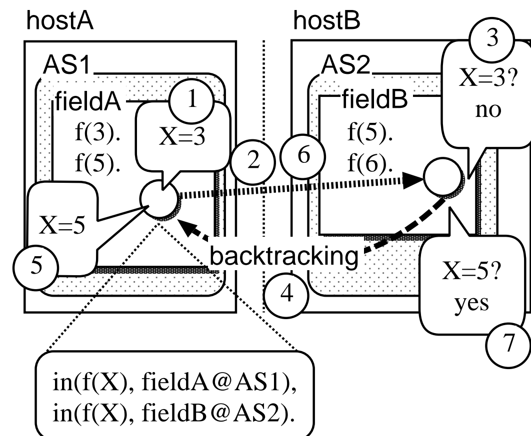

Figure 5. Backtracking and unification between two hosts.

4 Implementation

We have implemented Maglog on Java environment through extending PrologCafé[7] which is a Prolog-to-Java source-to-source translator system.

The program of an agent which is a set of Prolog clauses is translated into Java source code with our Maglog translator, and then it is compiled into Java classes with a Java compiler. As mentioned in Section 2.1, an agent can import Prolog clauses from a field at run time. These clauses are interpreted by the Prolog interpreter included in an agent instead of being compiled into Java classes. An agent runs as a thread in a process named agent server.

Agent servers have an XML-RPC interface which is accessible from applications written in any other language with support for XML-RPC.

The following operations from other systems are available through XML-RPC.

1. create and kill agents,
2. create and delete fields,
3. assert clauses into fields and retract clauses from fields,
4. get a list of names of fields,
5. get a list of IDs of agents currently exist.

Figure 6 shows a screen-shot of the user interface program for manipulation of agent servers. It can create/kill agents and create/delete fields, and can browse both contents of fields and outputs of agents.

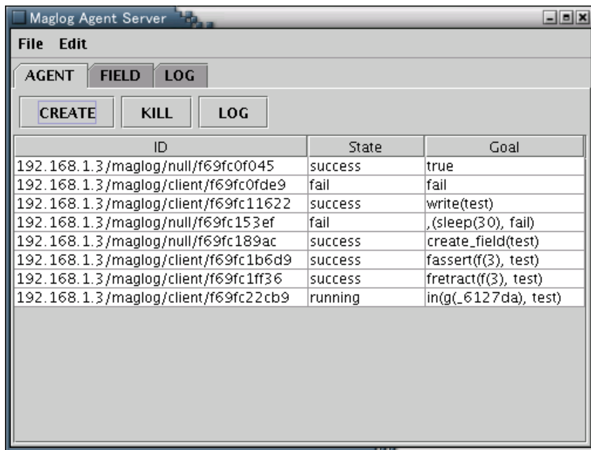

Figure 6. GUI for an agent server.

Implementation of Maglog features realized with the concept of the field is described in the rest of this section.

4.1 Predicate Library

As shown in Fig. 7, a field is implemented as a Java Hashtable, i.e. procedures in a field are put in a hashtable; a key is PredSpec of a procedure, and the value is a set of objects representing the procedure whose predicate indicator is PredSpec.

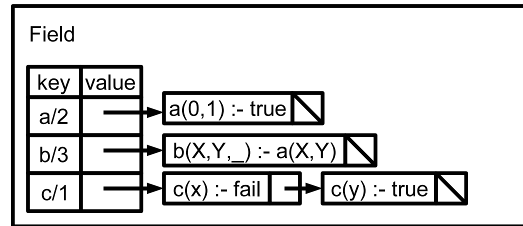

Figure 7. Structure of a field.

When an agent executes a predicate in `in/2`, it searches the predicate by specifying PredSpec from the hashtables of fields it currently enters and interprets the found value.

In order to improve execution rate, the concept of a static field is introduced into Maglog. It stores read-only procedures compiled into Java classes before the agent server to which the field belongs starts.

A static field is implemented as a Java Class Loader which receives PredSpec and loads the bytecodes of the class for the corresponding procedure.

According to experiments, an agent can execute a clause in a static field about 250 times faster than in an ordinary field.

4.2 Inter-agent Communication

As mentioned in Section 3.2, an agent which attempts to execute or to retract a non-existent clause in a field simply fails in the fail mode. While the agent in the block mode is blocked by calling the Java `wait` method.

When another agent adds one clause to a field, the blocked agents in the field are waked up by the Java `notifyAll` method and try to execute their goal. The agents whose target clause is added restart while the rest of the wake-up agents are blocked again.

4.3 Agent Migration

The migration of an agent is realized by using a Remote Procedure Call (RPC) as the following:

1. The source agent server encodes the agent as the argument of a RPC.
2. The source agent server gets serverID of the destination agent server from the second argument of the predicates `in/2`, `fasserta/2`, `fassertz/2`, and `fretract/2`.
3. The source agent server sends a RPC request to the destination agent for invocation of `receiveAgent` method.

4. The destination agent server decodes the argument of the RPC and restarts the decoded agent.

Two kinds of mechanisms for RPC are implemented: RMI and XML-RPC. RMI is more superior to XML-RPC from the viewpoint of the migration speed. On the other hand, XML-RPC is more firewall friendly than RMI. Because XML-RPC only uses port 80 while RMI uses two non-well-known ports. Users can choose the appropriate mechanism from them.

In order to reduce the traffic, a whole agent is not migrated initially. That is, Java classes compiled from Prolog predicates of an agent are transported on demand from the agent server on which the agent has been created.

5 Applications

In this section, two applications are demonstrated to confirm the effectiveness of the proposed model and Maglog.

1. Distributed e-Learning System[8][9]

   A distributed e-Learning system for asynchronous Web-based training has been built using Maglog. This system allows a student to study by himself in his own time and schedule, without live interaction of the teacher.

   Our distributed e-Learning system consists of content agents and user interface programs. Each content agent includes exercise data as well as teacher's functions to mark user's answers, to tell the correct answers and to show some extra information. Every computer of students receives some number of content agents from another computer when it joins the system and takes the responsibility of sending appropriate content agents to requesting computers. That is to say, while a student uses the system, its computer assumes a part of the system and plays both the role of the client and the server.

   Figure 8 is a screen-shot of the user interface program of this e-Learning system, which is developed in Squeak environment[10].

2. Schedule Arrangement System[11]

   This system arranges meeting schedule without human negotiations. It consists of negotiation agents and the user interface programs. Once a convener convenes a meeting through the system, agents move around the meeting participants and negotiate with them semi-automatically. This system corresponds to the proposed model as following.

   (a) Any user of this system can be a convener.

   (b) The number of computers participated in this system can be changed flexibly.

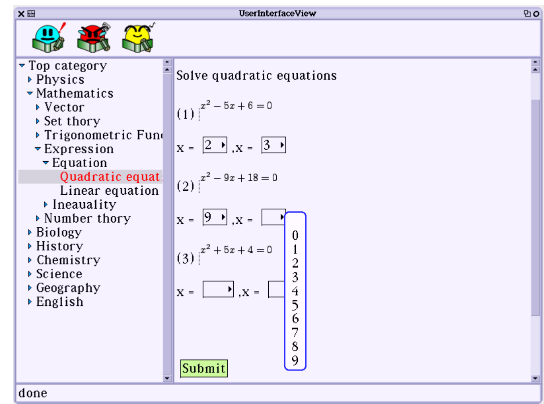

Figure 8. The e-Learning system.

   (c) Neither schedules of the participants nor the programs for negotiation are concentrated on a particular server. Instead, agents collect schedules of the participants and negotiate with them.

Figure 9 is a screen-shot of this schedule arrangement system written in Maglog and Java languages.

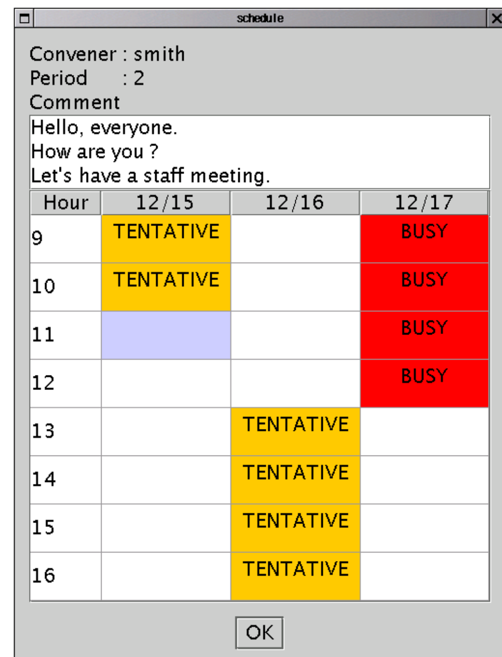

Figure 9. The schedule arrangement system.

6 Conclusion

A novel distributed model characterized by the following features is proposed in this paper.

1. Each computer has client functions concurrently with server functions assigned.

2. Every computer is able to join and to leave the system flexibly.
3. While server functions of the system do not change as a whole, they are shared on each computer flexibly. That is, the server functions on each computer vary according to the conditions of computers joined to the system.

To implement our model, the new framework named Maglog for mobile agent systems was designed and developed on Java environment. In Maglog, a concept called "field" is introduced. By using this concept, the following functions are realized.

1. Migration: Function that enables agents to migrate between computers.
2. Inter-agent communication: Indirect communication with other agents through the field. That is, an agent is able to import data or programs that other agents stored in the field.
3. Adaptation: Function that enables agents to execute programs stored in the field.

The effectiveness of the proposed model and Maglog are confirmed through the demonstrations of two applications: the distributed e-Learning system and the schedule arrangement system.

References

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

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

[3] Tarau, P.: Inference and Computation Mobility with Jinni, *The Logic Programming Paradigm: a 25 Year Perspective* (Apt, K., Marek, V. and Truszczynski, M.(eds.)), Springer, pp. 33–48 (1999).

[4] N, Fukuta. T, I. and T, S.: MiLog: A Mobile Agent Framework for Implementing Intelligent Information Agents with Logic Programming, *Proc. of the First Pacific Rim International Workshop on Intelligent Information Agents*, pp. 113–123 (2000).

[5] Kumeno, F., Ohsuga, A. and Honiden, S.: Flage: A Programming Language for Adaptive Software, *IEICE Transactions of Information & System*, Vol. E81–D, No. 12, pp. 1394–1403 (1998).

[6] Carriero, N. and Gelernter, D.: Linda in Context, *Communications of the ACM*, Vol. 32, No. 4, pp. 444–458 (1989).

[7] 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).

[8] Kawamura, T. and Sugahara, K.: A Mobile Agent-Based P2P e-Learning System, *IPSJ Journal*, Vol. 46, No. 1, pp. 222–225 (2005).

[9] Motomura, S., Kawamura, T., Nakatani, R. and Sugahara, K.: P2P Web-Based Training System Using Mobile Agent Technologies, *Proceedings of the 1st International Conference on Web Information Systems and Technologies*, pp. 202–205 (2005). Miami, USA.

[10] Ingalls, D., Kaehler, T., Maloney, J., Wallace, S. and Kay, A.: Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself, *Proceedings of ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 318–326 (1997).

[11] Kinosita, S., Kawamura, T. and Sugahara, K.: Mobile Agent based Schedule Arrangement System, *Proceedings of the 5th IEEE Hiroshima Student Symposium (HISS)*, pp. 205–206 (2003).