Implementation of a Logic-based Multi Agent Framework on Java Environment

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

Abstract— *We have proposed Maglog which is a framework for mobile multi-agent systems. Maglog is based on Prolog, and has the concept of field. A field is an object which can contain a knowledge base. With the concept of field, Maglog provides a simple and unified interface for 1)inter-agent communication, 2)agent migration between computers, and 3)utilization of data and programs on computers. In this paper, we present the implementation of Maglog on Java environment, in detail. Since we have implemented both command-line shell and GUI for Maglog, users can choose them for their needs. In addition, through XML-RPC interface for Maglog which we have also implemented, other systems can easily utilize Maglog.*

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 a multi-agent system, a number of autonomous agents cooperates mutually and achieves given tasks. Each agent is generated according to a given task and can have its own situation and operates under the situation. Situation consists of states and procedures where both of them can be dynamically changed in general. Therefore, it becomes necessary for the agent to dynamically hold the states and procedures (hereafter we refer them as a knowledge base). Moreover, when a number of agents are cooperating, it is necessary for them to share knowledge bases and to conduct knowledge communications between agents. In addition, mobility of agents is important in multi-agent system because of not only reducing network latency but simplifying architecture of software systems[2].

We have proposed Maglog [3, 4] which is a framework for mobile multi-agent systems. Maglog is based on Prolog, and has the concept of field. A field is an object which can contain a knowledge base. With the concept of field, Maglog provides a simple and unified interface for 1)inter-agent communication, 2)agent migration between computers, and 3)utilization of data and programs on computers.

In this paper, we present the implementation of Maglog on Java environment, in detail.

2. OVERVIEW OF MAGLOG

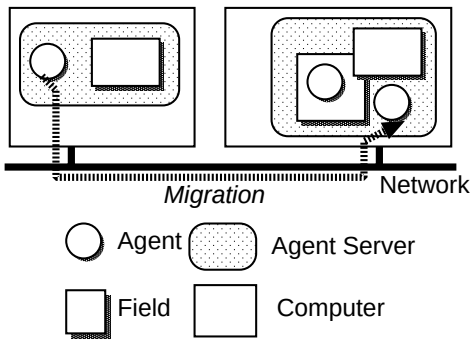

Figure 1 - Overview of a multi-agent system on Maglog.

Figure 1 shows the overview of a multi-agent system on Maglog. An agent runs as a thread in a process which we call an agent server. Mobile agents of Maglog are written in Prolog. Agent servers have objects which hold Prolog clauses. We call them fields. Builtin predicate `in(Goal, Field)` is for evaluation of a goal in a field. An agent can enter a field by this predicate. Entering a field, an agent can utilize data and programs in the field. With the concept of field, Maglog provides a simple and unified interface for 1)inter-agent communication, 2)agent migration between computers, and 3)utilization of data and programs on computers.

Inter-agent Communication

Agents belonging to the same field can be considered of forming a group. The knowledge within the field is shared by the agents. Moreover, by changing the knowledge within the field, agents can influence the actions of other agents.

An agent can communicate with other agents synchronously or asynchronously by reading/writing Prolog clauses from/into fields. Updating knowledge base in a field can be done by the following predicates.

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

The first argument `Clause` of these predicates is a clause to be added or deleted from the field specified by the second argument `Field`.

Through these predicates, agents can communicate with other agents also synchronously. An agent has three mode for execution of clauses included in a field. Firstly, in the error mode, an agent is halted permanently when it intends to execute a non-existent clause in a field. Secondly, in the fail mode, it is failed as an ordinary Prolog interpreter under the same condition. Finally, in the block mode, it is blocked until the target clause is added to the field by another agent. For agents in the block mode, a field can be used as a synchronous communication mechanism such as a tuple space in Linda model[5].

Figure 2 shows an example that an agent generates a child agent and waits until the child returns the result.

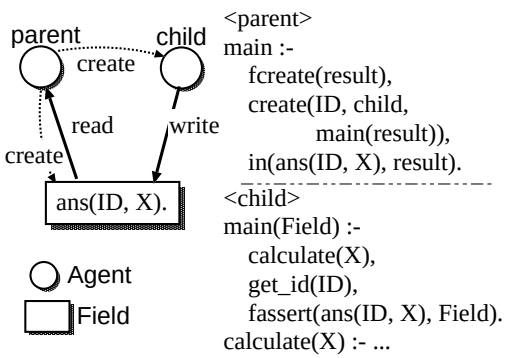

Figure 2 - Agents can communicate synchronously through a field.

Migration

The second arguments of the following predicates can be like `FieldName@HostAddress`.

```
in(Goal, Field)
fasserta(Clause, Field)
fasserta(Clause, Field)
fretract(Clause, Field)
fclause(Head, Body, Field)
```

If a host address specified, the agent will go to the host and access the field. Figure 3 shows that the agent unifies `f(X)` with clauses in two fields which are located in different hosts using `in/2`.

Dynamic Change of Behavior

When an agent enters into a field, it can use combined data and program of itself and the field. Therefore, an agent needs not to hold all the data and program by itself to solve a problem, but rather entering to appropriate fields which provide necessary data and program.

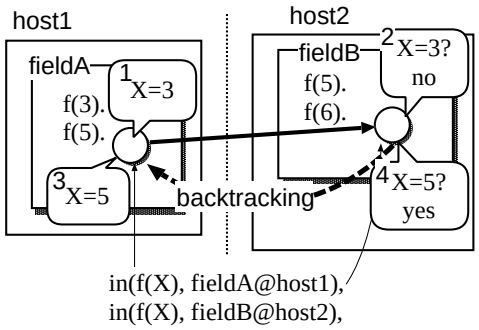

Figure 3 - Backtracking and unification between two computers.

Besides, an agent can change its behavior dynamically through entering a field. Figure 4 shows an example. 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.

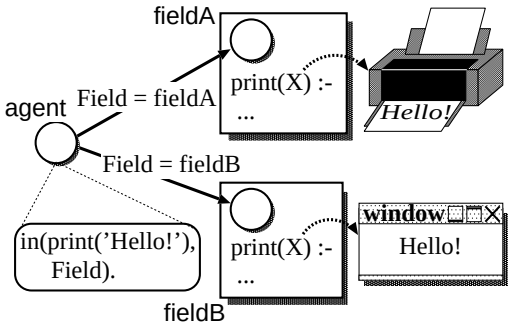

Figure 4 - Dynamic Change of agent's behavior through entering a field.

3. IMPLEMENTATION

We have implemented Maglog on Java environment through extending PrologCafé which is a Prolog-to-Java source-to-source translator system[6]. Both agents and after-mentioned static fields are translated into *.java files with our Maglog translator and then compiled into *.class files with a Java compiler.

Basic Components

In this section, we describe the Java implementation of agent, agent server, and field.

1. Agent

   Figure 5 shows the structure of an agent.

   (a) `ID` is given from the agent server when the agent is created by the server. An ID of an agent is composed of the IP address of the host in which the

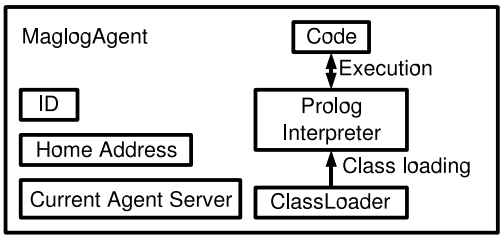

Figure 5 - Structure of an agent.

agent server runs and the time when the agent created, so that an ID is globally unique.

(b) `HomeAddress` is the address of the agent server in which the agent created.

(c) `CurrentAgentServer` is a reference to the agent server on which the agent currently runs. This reference is used to invoke methods of an agent server when agents want to access fields or migrate to other servers.

(d) `Code` is a predicate that the agent currently executing. Predicates are executed with a Prolog interpreter in an agent.

(e) `ClassLoader` is a serializable classloader which contains the classes of the predicates which are the knowledge and the program of the agent.

2. Agent Server

Figure 6 shows the structure of an agent server. An agent server creates an agent by `createAgent` method. `createAgent` method creates an agent from the agent repository, which contains classes of predicates of agents, and pushes into the agent scheduler. The agent scheduler has threads for agents. When `sendAgent` method invoked, the agent server stops the agent and removes it from the agent scheduler and sends to the destination.

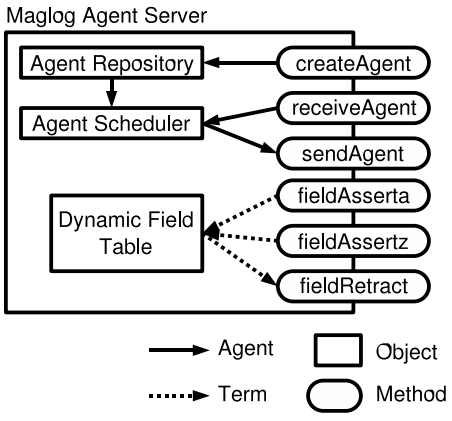

Figure 6 - Structure of an agent server.

3. Field

We have implemented two kinds of field, dynamic field and static field. Static fields are pre-compiled predicates. Therefore agents cannot modify static fields, however executions of predicates in static fields are relatively fast. Executions of predicates in dynamic fields are relatively slow however agents can assert and retract clauses in dynamic fields.

Figure 7 shows the structure of a dynamic field.

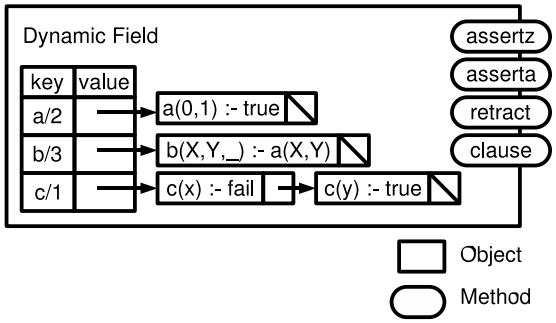

Figure 7 - Structure of a dynamic field.

Clauses in the dynamic field are put in a hashtable. Keys of the hashtable are predspec of clauses in the value. Dynamic Field has `assertz`, `asserta`, `retract` and `clause` method. Both `assertz` and `asserta` are used for addition of a clause. The only difference is the position of the added clause. The former adds the clause at the bottom of the predicate, while the latter inserts the clause at the top of the predicate. These methods are synchronized method and `retract` and `clause` call `wait` method and `assertz` and `asserta` call `notifyAll` method.

Core Builtin Predicates

1. in/2

`in/2` can be used in nested form, like `in(in(Goal, FieldA), FieldB)`. At first, the implementation of `in/2` makes a list of fields. And goes to the host in which the field in the list exists and check whether any clause that can be unified with `Goal` exists in the field. This procedure iterates the list from the inside of the nested `in` until an unifiable clause is found. If it is found, the agent evaluates the goal with the clause and returns to the initial host. If no clause is found in all nested fields, the agent evaluates the goal in the agent's own knowledge base.

2. fasserta/2, fassertz/2

An agent executes `fasserta(Clause, Field)` goes to the host in which the field exists and calls the `asserta` method of the field with the `Clause` in the argument. Then the agent returns to the initial host. `fassertz(Clause, Field)` works in the same manner.

3. fretract/2

An agent executes `fretract(Clause, Field)` goes to the host in which the field exists and calls the `retract` method of the field with the `Clause` in the argument. If any clause which can unify with `Clause` is not found in the field and the agent is in block mode, the agent stops and waits for the clause which can unify with `Clause`. If the agent is in fail mode, it fails immediately. If the unifiable clause is found, the agent removes it and returns to the initial host.

4. fclause/3

   `fclause(Head, Body, Field)` works in the same manner as `fretract/2` however `fclause/3` calls the `clause` method of Dynamic Field and doesn't delete clauses, just reads clauses.

User Interface

There are two user interface to manipulate agent servers, Command-line shell (hereafter we refer to it as CUI) and graphical user interface (hereafter we abbreviate as GUI). Both of them can create and delete agents and fields, and can browse contents of fields and outputs of agents.

1. GUI

   Figure 8 shows GUI of the agent server. This user interface is implemented with Swing toolkit. The interface extends the class of agent server, i.e the interface is an agent server, so that the list of agents, the list of names of fields, the status of agents and contents of fields are updated in real-time.

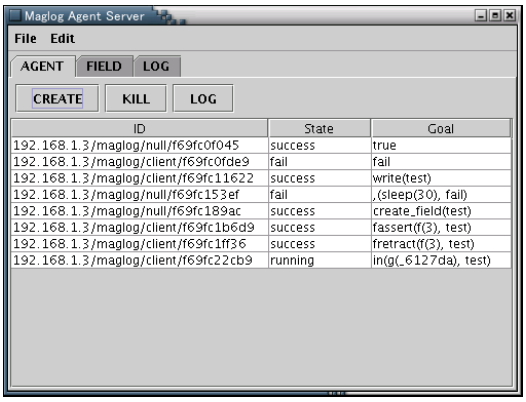

Figure 8 - GUI for an agent server.

2. CUI

   In addition to GUI, We have implemented CUI as a command named maglogsh which is a command-line shell to manipulate the agent server. Table 1 lists builtin commands of maglogsh.

   The maglogsh command runs on a Java virtual machine independent of the Java virtual machine on which an agent server runs. And maglogsh invokes the methods of the agent server via Java RMI. Therefore, maglogsh can access agent servers on remote hosts.

Table 1 - Builtin commands of maglogsh.

Command	Description
connect	connects to the agent server
ls	lists the names of dynamic fields in the agent server
cat	shows the content of the dynamic field
rm	deletes the field from the agent server
ps	lists agents in the agent server
show	shows the output of the agent
kill	deletes the agent
create	creates an agent

Integration with Other Systems

We have implemented a XML-RPC interface for the agent server using Marquée XML-RPC, an XML-RPC library for Java. XML-RPC is a simple remote procedure calling protocol using XML as the encoding format[7]. Therefore, any programming languages which supports XML-RPC can access Maglog agent servers.

Through XML-RPC, other systems can do the following operations.

- create and kill agents.
- create and delete dynamic fields.
- assert and retract clauses in dynamic fields.
- get a list of names of dynamic fields.
- get a list of IDs of agents.

Prolog clauses in return values and arguments of requests are translated to data types of XML-RPC by the agent server.

APPLICATIONS

We will outline several applications developed using Maglog.

1. e-Learning System[8]

   An P2P-based e-Learning system has been built using Maglog. This e-Learning system has two distinguishing features. Firstly, it is based on P2P architecture for scalability and robustness. Secondly, each content in the system is not only data but an agent so that it can mark user's answers, tell the correct answers, and show some extra information without human instruction. Maglog plays an important role to realize the both features. Figure 9 is a screen-shot of the user interface program of this e-Learning system, which is developed in Squeak environment[9]. This e-Learning system consists of about 2,000 lines of Maglog code and about 4,000 lines of Squeak code.

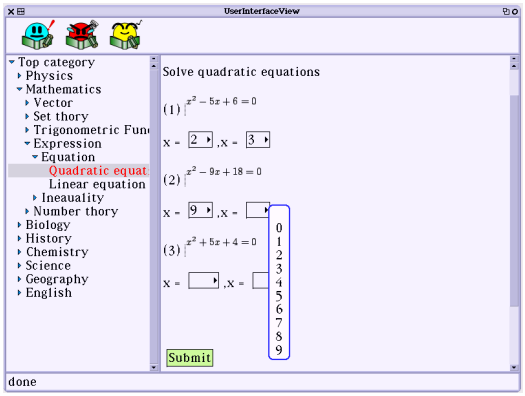

Figure 9 - The e-Learning system.

2. Schedule Arrangement System[10]

   This system, which has been developed using Maglog, establishes and arranges meeting schedule without human negotiations. Once a convener convenes a meeting through the system, an agent moves around the members of the meeting and negotiates with them automatically. Figure 10 is a screen-shot of this schedule arrangement system which consists of about 400 lines of Maglog code and about 4,000 lines of Java code.

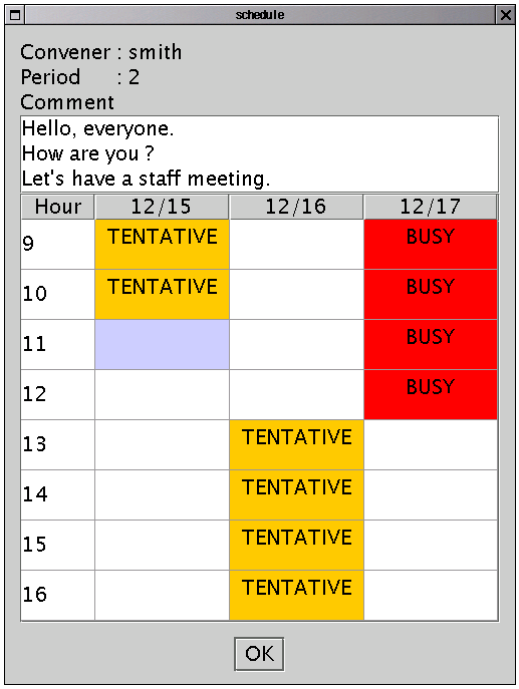

Figure 10 - The schedule arrangement system.

3. HECS System[11]

   Maglog is used for HECS (Heterogeneous Constraint Solving) system, which was supported in part by IPA (The Information-technology Promotion Agency) under grant of 2003 Exploratory Software Project, to coordinate distributed solvers each other.

Related Works

There are several mobile agent frameworks realized as a set of class libraries for Java such as Aglets[12] and MobileSpaces[13]. The combination of one of them and a Prolog interpreter/compiler written in Java such as NetProlog[14] and Jinni[15] have some similarity to Maglog. The main difference between the combination and Maglog is the class of mobility. Their mobility is so-called weak mobility, in which only its clause database is migrated. On Maglog, all of the execution state including execution stack can be migrated (so-called strong mobility), therefore agents on Maglog can backtrack and unify variables during migration. That makes programs on Maglog simple and understandable.

Conclusion

In this paper, we presented the implementation of a mobile agent framework Maglog, in detail. With the concept of field, Maglog provides a simple and unified interface for 1)inter-agent communication, 2)agent migration between computers, and 3)utilization of data and programs on computers. Since we have implemented both CUI and GUI for Maglog, users can choose CUI or GUI for their needs. And through XML-RPC interface which we have implemented, other systems can easily utilize Maglog. Through XML-RPC interface, several applications have been developed.

References

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

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

[3] Kawamura, T., Kinoshita, S., Sugahara, K. and Kuwatani, T.: A Logic-based Framework for Mobile Multi-Agent Systems, *Proceedings of International Conference on Integration of Knowledge Intensive Multi-Agent Systems* (Hexmoor, H.(ed.)), pp. 754–759 (2003). Boston, Massachusetts, USA.

[4] Kawamura, T., Kinoshita, S. and Sugahara, K.: Implementation of a Mobile Agent Framework on Java Environment, *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems* (Gonzalez, T.(ed.)), pp. 589–593 (2004). MIT, Cambridge, USA.

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

[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] Winer, D.: XML-RPC Specification, http://xmlrcp.com/spec.

[8] Kawamura, T., Kinoshita, S. and Sugahara, K.: A Mobile Agent-Based P2P e-Learning System, *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems* (Gonzalez, T.(ed.)), pp. 873–877 (2004). MIT, Cambridge, USA.

[9] 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).

[10] 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).

[11] Banbara, M., Tamura, N., Inoue, K., Kawamura, T. and Tamaki, H.: Java Implementation of a Distributed Constraint Solving System, Exploratory software project, Information-technology Promotion Agency Japan (2003).

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

[13] 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).

[14] de Carvalho, C. L., Pereira, E. C. and da Silva Julia, R. M.: NetProlog: A Logic Programming System for the Java Virtual Machine, *Proceedings of the 1st International Conference on Enterprise Information Systems*, pp. 591–598 (1999). Setubal, Portugal.

[15] 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).