GUARDNET: A DISTRIBUTED AND CONCURRENT PROGRAMMING ENVIRONMENT FOR MULTI-AGENT SYSTEMS

Motoyuki Takaai, Hideaki Takeda and Toyoaki Nishida
Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-01 Japan
Phone:+81-7437-9-9211 ext.5316, FAX:+81-7437-2-5269
Email:{motoyu-t,takeda,nishida}@is.aist-nara.ac.jp

ABSTRACT

GuardNet is a programming environment that supports distributed and concurrent development of multi-agent systems. Since agents in multi-agent systems are distributed and work concurrently, they are often developed distributively and concurrently. Because GuardNet itself is a multi-agent system, GuardNet provides the following three services to support programmers developing such multi-agent systems; (1)exchanging agent specifications instead of agent programmers, (2)generation of agent templates from exchanged specifications, and (3)proxing agents for incomplete or under-programming agents. In this paper, we describe GuardNet's architecture, mechanism of these services, and practice of agent programming with GuardNet.

1. INTRODUCTION

In recent years, researchers are interested in problem solving and knowledge information processing by multi-agent systems. In multi-agent systems, a number of computer programs (so called *agents*) are connected to each other, so that larger and more complex problems can be solved. For example, we have proposed Knowledgeable Community as a framework of knowledge sharing and reuse, which is based on multi-agent systems [Ni94][Ta95]. Through out experience, we have realized that programming of multi-agent systems is not easy task.

It is because agents in multi-agent systems are distributed and work concurrently. In this paper, we show a new programming environment called GuardNet that supports distributed and concurrent development of multi-agent systems.

GuardNet generates a special support agent called Guardant for each agent to be programmed, and each Guardant supports the programmer for programming the original agent by communicating her/him and other Guardants. Guardants provide the following three services to support programmers of multi-agent systems; (1) Exchanging agents specifications instead of agent programmers, (2) Generation of agent templates from exchanged specifications, and (3) Proxing agents for incomplete or under-programming agents.

The following sections are organized as follows. In Section 2, we show our basic approach to model agents, i.e., agent as virtual knowledge base. In Section 3, we show the architecture of GuardNet and Guardant. In next three sections, we show Guardant's three functions. In Section 7, we show GuardNet implementation and examples of GuardNet programming. In Section 8, we show how GuardNet supports programmers by using a test programming to build a simulation system of a multiple robot system. Section 9 summarizes the paper.

2. AGENT AS VIRTUAL KNOWLEDGE BASE

In this paper, we employ KQML(Knowledge Query and Manipulation Language)[Fi92] as communication protocol among agents. In KQML, agents are interpreted as virtual knowledge base (VKB), that is, each agent is expected to behave as knowledge base, even if they are not knowledge bases actually. It means that they accept messages of knowledge operation (query, deleting, addition etc.).

A KQML message includes message intention, sender's and receiver's names, name of knowledge representation language and so on. Intention is represented as *performative*, that is defined as the operations of knowledge base (i.e. query, reply, insert and delete information etc.).

Figure 1 shows the examples of KQML messages. `ask-if` means that the message is a query, `:content` means that the next is knowledge content, `:language` means that the next is name of knowledge representation language.

```
(ask-if :content (and (temple ?t todaiji)
                      (tel-number ?t ?tel))
        :language kif)
```

(a) Question Message

```
(reply  :content (and (temple t1 todaiji)
                      (tel-number t1 ''12-3456''))
        :language kif)
```

(b) Answer Message

Figure 1: Examples of question and answer message

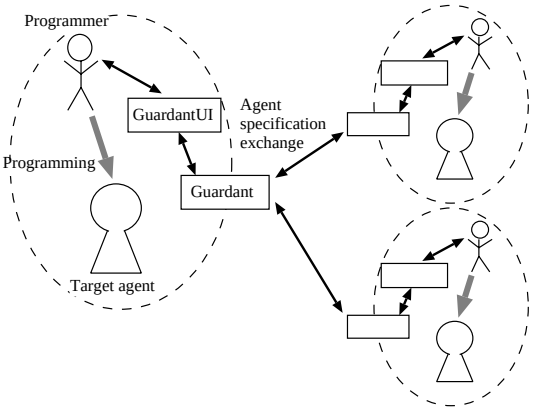

Figure 2: Connections among developers by GuardNet

3. SUPPORTS OF MULTI-AGENT SYSTEM DEVELOPMENT BY GUARDNET

We adopt a multi-agent system as the architecture of GuardNet. Since multi-agent system is used to build flexible and extensive systems, programming environments of multi-agent systems should be also flexible and extensive. Then it is natural to build the programming environment by a multi-agent system.

GuardNet provides a special support agent called Guardant and an editor-embedded agent for each programming agent (see Figure 2). We call the programming agent the *target* agent for the Guardant. We also call the other programming agents *outer* agents for the Guardant.

A Guardant support programmers for programming the target agent by communicating programmers via editor-embedded agents and programmers of outer agents via other Guardants.

The network of GuardNet is independent from the network of the developing agent system, and specifications of programming agents are exchanged agents on this network.

Figure 3 shows the architecture of Guardant. Guardant is not only the agent of GuardNet but also the agent of the developing agent system.

Specification manager supports cooperation among programmers during the decision processes of message specifications. The message specifications which are decided or under consideration are recorded to the target agent specification database. Agent proxing is realized by this database and outer agent specification database.

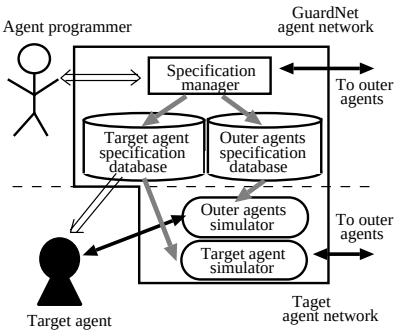

Figure 3: The architecture of Guardant

```
(ask-if :content (and (temple ?t $name)
                      (tel-number ?t ?tel))
        :language kif)
```

(a) Message Class

```
($name = todaiji)  => (?t = t1, ?tel = "12-3456")
($name = saidaiji) => (?t = t2, ?tel = "23-4567")
```

(b) Message Instance

Figure 4: Examples of Message specifications

4. DECISION OF MESSAGE SPECIFICATIONS BY GUARDANT

Since one of the important features of agents is to manage messages, that is exchanging various messages with other agents, GuardNet supports the decision of *message specification* as specification of agents.

A programmer of an agent must discuss with other agent programmers, because the agent exchanges various messages with other agents. GuardNet supports this process of decision by exchanging requests of message specifications among Guardants instead of these programmers. This exchanging of requests are controlled by state transition diagram of conversation for action.

The benefits of this approach are as follows;

Message specifications are clearly defined: Decision by human communication is often vague and uncertain. In this approach, each specification is explicitly represented, and certainty is verified by state transition diagram for action.

Message specifications are re-usable: Message specifications are recorded in the database, so that we can use them in generating agent templates and proxing agents.

4.1 Message specification

Message Specifications are either message classes or message instances which are defined as follows;

Message Instance: message itself represented by message class name and substitution for variables

Message Class: abstract representation of message instance using variables

Message instance can be made from message class substituting constants for the variables. Guardant exchanges message classes for the decision of message specifications. Figure 4 shows examples of the message specifications which use KQML as message protocol and KIF [Ge90] and knowledge representation language.

4.2 Databases of message specifications in Guardant

Guardant have two databases. One is target agent specification database where the message specifications of the target agent are stored. The other is outer agent specification database where the message specifications that the programmer requests to other programmers are stored.

4.3 The process of manage specifications of decision

We show processes of the decision of message specifications as follows;

Here we assume that programmer U of agent A uses Guardant G, and U' of A' does G' too.

(1) U registers a request with the outline texts to G.
(2) The process of decision of message class.
(2-a) U inputs a draft of the message class to G.
(2-b) G sends the draft to G'.
(2-c) G' shows the draft to U'.
(2-d) U' inputs the reply (acceptance or counter-offer) for it to G'.
(2-e) G' sends the reply to G.
(2-f) G shows the reply to U.
(3) G and G' record draft and accepted (or promised) message class to their database.

In the process from (2-a) to (2-f), specification manager in Guardant runs according to state transition diagram of conversation for action [Wi88] until U and U' agree the specification. During this process, the conversations are recorded to their database by Guardants. Since Guardant prepares state transition diagram of conversation for action for each message specification, programmer can handle a number of processes of decision of some message classes at once.

4.4 Addition of message instances

Programmers can add message instances to decided message specifications of target agent. They are stored in target agent specification database with the message class in the Guardant.

Programmers can also do to the message specifications of outer agents, and they are stored in outer agent specification database.

Guardant can import the message instances related to the target agent from other Guardants.

5. SUPPORT OF THE AGENT CODING BY GUARDNET

GuardNet supports agent coding by using database of the message specifications.

```
(defagentkc foo
 :mbus-port 2392
 :message-handler 'foo-handler)
(setq foo-handler
 '((((:performative . (?or ask-if))
     (:kif-content . (and (temple ?t $name)
                          (tel-number ?t ?tel)))
     (:language . kif)
    ) . do-foo-telnumber)
  (((:performative . (?or ask-if))
    (:kif-content . (and (temple ?t $name)
                         (open-time ?t ?o-time)))
    (:language . kif)
    ) . do-foo-open-time)))
(defun do-foo-telnumber (env)
"Title: An phone number of a temple
Comment: phone number is local number)"
  (let ((binding *kif-binding*))
  ))
(defun do-foo-open-time (env)
"Title: Admission time for visiting
Comment: $name is a temple name."
  (let ((binding *kif-binding*))
  ))
```

Figure 5: The interface part of an agent generated by Guardant

We developed an agent programming language called KC-Basic which runs on Common-Lisp. In KC-Basic, the message receiving parts of agent programs consist of the following elements;

(1) agent registration
(2) registration of message classes and functions to execute for them
(3) the bodies of the functions

When an agent receives a message of which pattern matches with one of the message classes in (2), the associated function for it is called and executed along with its body written in (3).

Guardant makes the template of this message receiving parts of agents the agent automatically from the massage specifications. Figure 5 shows the template from the message specifications in Figure 4. When the programmer modifies the function body of *do-foo-telnumber* and *do-foo-open-time*, this agent will be executable.

6. AGENT PROXING

Agent proxing by GuardNet is that Guardants works as proxies of programming agents in case that there does not exist necessary agents yet or exist only incomplete agents which are not good for use yet. Agent proxing can make the simulation of such incomplete agent systems. We realized the agent proxing by using specifications from message exchanging explained in Section 4.

6.1 Types of agent proxing

We provide two types of agent proxing.

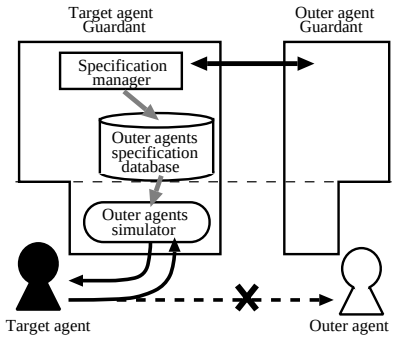

Figure 6: Outer Agent Imitating by Guardant

Agent proxing for outer agents: For verifying execution of the target agent, Guardant becomes proxy agents for outer agents to which the target agent wants to communicate. If some of the outer agents which the target agent want to communicate are not at work, we cannot verify the execution of target agent. By proxing such agents, it becomes possible to check the execution of target agent.

Agent proxing for the target agent: When the target agent is not good for use (for example in debugging), Guardant becomes a proxy agent for the target agent. If other agents want to communicate to the target agent, the Guardant responds to it instead of the target agent.

6.2 Type of message actions to simulate

We can identify the following four types of message actions that proxy agents should simulate as the normal agents do.

Ask: Send the query message and expect to get the answer

Reply: Send the answer for the query message

Tell: Send the message to tell information

Listen: Receive message to get information from outer agents

For example, simulation of reply action is as follows;

(1) Find specifications which match the incoming messages in the specification database.

(2) Compose reply messages by using the query message and the found specifications.

6.3 Agent proxing mechanism

In agent proxing for outer agents, Guardant prepares a simulator for each of them. The simlulators react as knowledge database which have the message instances of outer agents' specification database in Guardant(see Figure 6).

In agent proxing for target agent, the simulator react as knowledge database which have the message instances of target agent's specification database in Guardant.

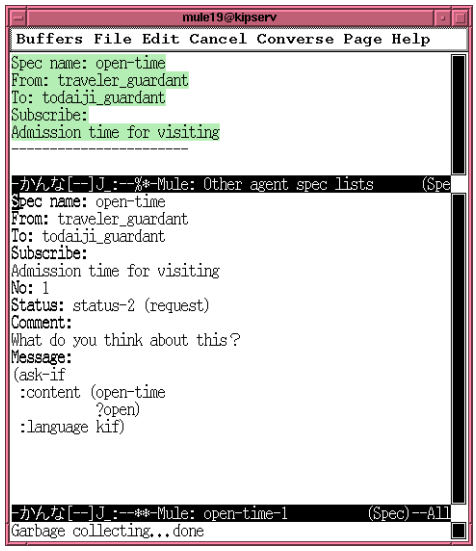

Figure 7: The developer of the traveler agent exchanges agent specifications with the other developer by Guradant

6.4 Control of the message flow

To proxy agents, it is necessary to control the message flow so that exchange of message with agent's proxy agents becomes possible.

When outer agents are already proxied, all messages from the target agent to the outer agents are received by Guardant and the real outer agents never receive these messages. Guardant plays the outer agents' roles and uses this agents' names for answer.

In the multi-agent system that uses "federation architecture"[Pa92], all messages are sent to a special agent called *facilitator* first. We use the facilitator to realize control of these message flows.

7. IMPLEMENTATION OF GUARDNET

7.1 GuardNet architecture

GuardNet consists of two types of agents, i.e., Guardant and GuardantUI.

Guardant continues working whether programmer is using Guardant or not, to wait for the messages from other Guardants always. Guardant manages all the data of message specifications. Guardant is implemented on GNU Common Lisp.

GuardantUI interacts with a programmer for Guardant. GuardantUI is implemented on Emacs Lisp (ver 19) and is embedded to the editor that programmer uses. A programmer can start and exit GuardantUI anytime.

7.2 Examples of system's execution

Here we show an example to illustrate how GuardNet works. In this example, there are an agent to represent a traveler, and some agents to represent temples.

Figure 7 shows a snapshot of GuardantUI, where the programmer of the traveler agent is asking to the

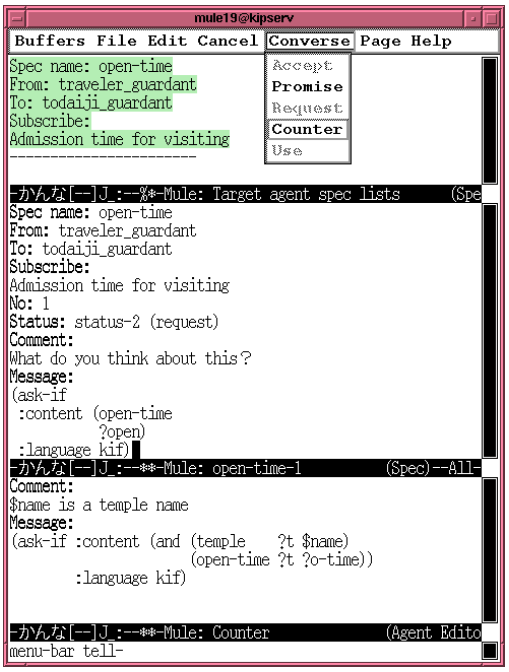

Figure 8: The developer of Todaiji-temple agent exchanges agent specifications with the other developer by Guradant

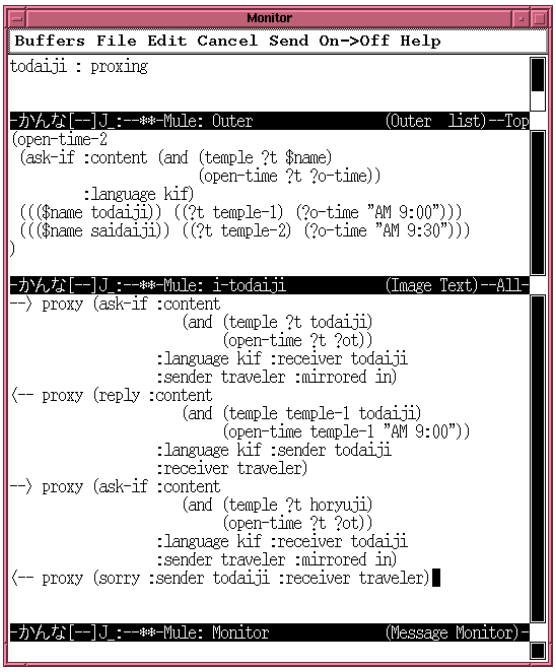

Figure 9: The developer of the traveler agent makes imitations of other agents by Guradant

programmer of Todaiji temple agent about possible time to visit the temple. The upper window shows the list of message specifications which are registered up to now. The specification about the information of the possible time to visit the temple is chosen and the contents of exchanging message specifications is shown in the lower window. The message specifications that the traveler agent programmer requests to Todaiji temple agent programmer are the same to those shown in Figure 4. The requested content for it is sent to the Guardant of Todaiji temple agent.

Figure 8 shows a snapshot when Guardant of Todaiji temple agent received the request, and replied a counter-offer. The upper window shows the list of message specifications which are registered up to now. The middle window shows the contents of the received requirement.

Figure 9 is a snapshot when the traveler agent programmer is exchanging the message with the todaiji agent which is currently proxied by the traveler agent's Guardant. The upper window shows that Todaiji temple agent is now in agent proxing. In the middle window, there is data of Todaiji temple agent's message specifications (see Figure 4), and the admission time for visiting Todaiji temple and Saidaiji temple are given as message instances. The lower window shows how execution of the traveler agent works with the proxy agent of Todaiji temple agent. Instead of Todaiji temple agent, Guardant of the traveler agent receives the message which was sent from the traveler agent to Todaiji temple agent and it replies a message to the traveler agent.

8. EXPERIMENTS ON AGENT PROGRAMMING BY GUARDNET

We used GuardNet to build a simulation system of a multiple robot system (see Figure 10) to show how it can support programmers of multi-agent systems.

There are the following three tasks to simulate.

- Robots bring the object to the user which he/she wants to get.
- Robots put objects in an area to the rack in order.
- Robots deliver objects.

We have two autonomous mobile robots (one has hands), an autonomous rack and an automatic door. In this test, we asked two programmers to build the system.

8.1 Meeting among programmers

Before decision of the agent programming by GuardNet, the programmers had a meeting. In the meeting, they talked what agents they would make, and what information the agents exchange. In this case, they decided the architecture of the agent system, i.e., four agents for robots, a user-interface agent and a planning agent. One programmer took charge of four agents, the other took charge of two (see Figure 10).

8.2 The experiments on the decision of message specifications by Guardant

They made 25 messages. 8 messages in them are just re-used of the other messages, so that they have no decision processes. In the decision processes of 9 messages, there are at least one counter-offer. The other messages are accepted without any counter-offers.

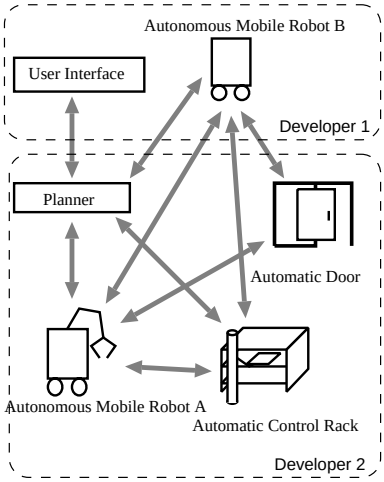

Figure 10: An example of agent system on physical world

The later the experiments proceeded, less counter-offers were made. It seems that the programmers formed representation patterns of message specifications.

The counter-offers can be classified into 6 groups as follows;

- Point out mistakes
- Modification of representation
- Addition of representation and information
- Fundamental change of representation
- Emphasis on his own opinion again
- Query

We can conclude the following advantages of GuardNet from this experiment.

- Programmers can communicate and decide the agent specifications, regardless their physical distances.
- Each programmer can manage agent specifications concurrently.
- The decision process of the agent specifications are done in a relatively short time.
- Even if a programmer is not at work, Guardant can provide the agent specifications to other programmers instead of the programmer.

But we also obtained the following disadvantages;

- Many (more than about 5) parallel decision processes of message specifications often make the programmer confuse.
- Programmers may want to unify expressions of different messages, but GuardNet don't support it.

9. CONCLUSION

By management of message specification, GuardNet supports decision of message specification, agent coding and debugging. To share the expression of message expression, GuardNet needs the function to manage ontology. In the KSE they study collaborative ontology construction on world wide web [Fa94].

Parman [?] is intelligent parametric design tool supporting collaborative engineering. The agent of Parman consists a design tool and a human user, and the relation of them is similar to our Guardant and developer.

10. SUMMARY

In this paper we described a multi-agent system GuardNet which supports cooperative development of multi-agent systems. It provides a new environment for cooperative, distributed, and asynchronous development of multi-agent systems.

REFERENCES

[Ni94] T. Nishida and H. Takeda: Towards the Knowledgeable Community, In K. Fuchi and T. Yokoi, editors, Knowledge Building and Knowledge Sharing, Ohmsha, IOS Press, 1994, pp.155–164.

[Fi92] T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, P. Pelavin, S. Shapiro, and C. Beck: Specification of the KQML agent-communication language, Technical Report EIT TR 92-04, Enterprise Integration Technologies, 1992.

[Wi88] T Winograd: A language / action perspective on the design of cooperative work, In Irene Greif, editor, Computer-Supported Cooperative Work: A Books of Reedings, Morgan Kaufmann, 1988.

[Ta95] H. Takeda and K. Iino and T. Nishida: Agent Organization and Communication with Multiple Ontologies, the International Journal of Cooperative Information Systems (IJCIS),1995.

[Ge90] M. R. Genesereth and R. E. Fikes: Knowledge Interchange Format version 3.0 reference manual, Technical Report Logic-90-4, Computer Science Department, Stanford University,1990.

[Pa92] R. S. Patil, R. E. Fikes, P. F. Patel-Schneider, D. McKay, T. Finin, T. R. Gruber, and R. Neches: The DARPA knowledge sharing effort: Progress report, In Charles Rich, Bernhard Nebel, and William Swartout, editors, Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference. Morgan Kaufmann, 1992.

[Ku94] Daniel Kuokka and Brian Livezy: A Collaborative Parametric Design Agent, Proceedings of the AAAI-94,pp.387-393

[Fa94] A. Farquhar, R. Fikes, W. Pratt, and J. Rice: Collaborative Ontology Construction for Information Integration, http://www-ksl.stanford.edu/KSL_Abstracts/KSL-95-63.html