SAGE and Its Application to Inter-company EC

Ryusuke Masuoka, Tamami Sugasaka, Akira Sato,
Hironobu Kitajima and Fumihiro Maruyama
NetMedia Laboratory
Fujitsu Laboratories Ltd.
2-2-1 Momochihama, Sawara-ku, Fukuoka 814, Japan
{masuoka, tamami, satoyan, kitajima, superb}@flab.fujitsu.co.jp

Abstract

We are doing research under the SAGE (Smart AGent Environment) project on seamless integration of information distributed over networks like the Internet. We avail ourselves of conversational agents speaking ACL (Agent Communication Language) as an enabling technology for such use of distributed information. We have applied SAGE to an inter-company EC (Electronic Commerce) prototype, which is planned to be deployed for real services. We have named this prototype SAGE:Francis.

Through the application process, we have agentified legacy software like database management systems (DBMSs) as database agents and also agentified users as user agents into SAGE. Then we put mediator agents called facilitators to help the inter-operation through ACL messages between these agents. We have identified useful functions of facilitators and implemented some of them. With the agentification and functions of facilitators, we have made it possible to integrate separate and disperse databases using different ontologies (terminologies).

We report, in this paper, inter-company EC as an application area for SAGE, identified functions of facilitators, implementations of the agents, the internal workings of the system and its performance.

1 Introduction

We report, in this paper, SAGE and its application to inter-company EC (Electronic Commerce). The prototype, which is named SAGE:Francis, is planned to be deployed for real services.

With the advent of WWW and the exponential growth of the Internet, immense amount of information is stored on the network. Still it is very hard to find what one really wants. One can use search engines on the network, but this often produces too long listings for human beings to parse. More recently, pages are loaded with "multimedia" stuff. Human intervention is a must to parse such pages.

Information one needs is somewhere on the network. But if one can not find it, it is as same as nonexistence for him or her.

To address such issues, we are working on SAGE (Smart AGent Environment) project. This project utilizes software agent technology, especially of those conversational agents

which communicate by ACL (Agent Communication Language) [1]. SAGE should provide an infrastructure for those agents.

Main research areas for SAGE project include:

- Agentification of users and legacy applications
- Facilitation of interoperation between agents by mediator agents (called facilitators)
- Message formats and transactions
- Libraries and tools for the above
- Real-world applications

We chose inter-company EC to try SAGE out for first, rather than consumer EC and other application areas because of the following reasons. The first reason is that enterprises can use resources to have useful systems. The second reason is that a limited environment of inter-company EC makes first deployment easy. After a successful deployment, which will lower the resources and will make it possible to provide higher functions, SAGE will be applied to less limited environments such as consumer EC.

In inter-company EC, there are many stages such as search for products, price negotiation, procurement and payment. There are several enabling technologies in the market including WitWeb from Fujitsu [8], TWX-21 from Hitachi [9] and TRADE'ex from TRADE'ex [10]. These systems use a centralized server and provides services there. As the size of system grows, however, it is necessary to be able to have distributed servers. There might also be demands on combining these systems together.

We believe SAGE can provide solutions for such situation. With SAGE, it is possible to integrate disperse data sources and to reuse legacy information sources such as databases. Especially with translation service provided by facilitator, it should be much easier to make data sources of different origins interoperate.

2 SAGE and Facilitators

We outline SAGE in this section. Implementation details are described in section 3.

As mentioned earlier, SAGE bases itself on conversational software agents. One needs common communication protocol, which can communicate messages in a structured way in order to convey "meaning." We decided to use Agent Communication Language (ACL) which consists of KQML (Knowledge Query and Manipulation Language) [5] and KIF (Knowledge Interchange Format) [4] for the common communication protocol.

Agents can be characterized by the roles assigned to those agents. We have user agents, database agents and mediator agents called facilitators in SAGE. User agents represent users as agents in SAGE. Database agents give databases agent interfaces. Facilitators stand between other agents and help those agents interoperate. Useful functions of facilitators for the purpose above are identified and described in section 2.1. Those agents can be arranged in a variety of ways. Figure 1 shows the arrangement of agents in our inter-company EC application, SAGE:Francis. This architecture is known as virtual catalog [3].

We have implemented each agent as physically separate. In some cases of agent implementations such as Infomaster of Stanford University [7], they are logically separate,

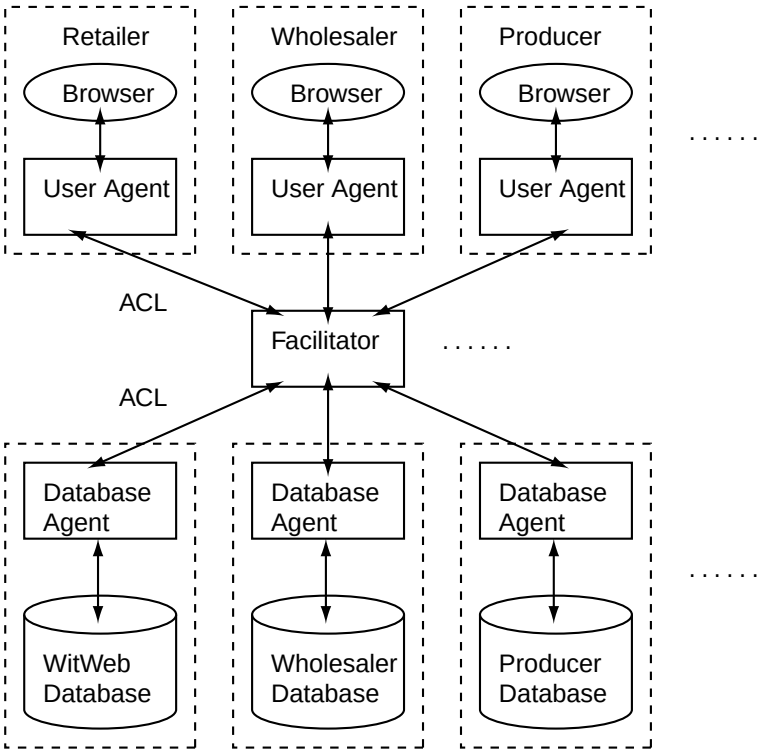

Figure 1: Architecture of SAGE:Francis

but physically in one process. This separation gives us great flexibility of the design and advantages in application of our agent systems. For example, it will alleviate managing complexity for the case where managing organizations are different for each agent.

2.1 Functions of Facilitators

We describe functions of facilitators identified as useful in this section.

Facilitators in our system assume a very important role of brokerage of messages by providing useful functions. Possible functions for facilitators include the followings:

- Delegation based on:
  - Advertisement
    - Category
    - Agent capability
    - Affiliation
    - Accessibility
  - Observation
- Translation
- Merging multiple messages
- Sorting the contents of messages

Delegations can be done through advertisement and observation. Service providers can advertise their capabilities or restrictions to facilitators. For example, service provider agents can advertise categories and message formats they can handle, their affiliations [1] and their accessibilities [2]. Facilitators use such advertisements as criteria for their choice of delegations.

Observing the messages sent back to facilitators in response to the messages from facilitators, facilitators can learn about corresponding agents. This information can be used for choice of delegations.

Facilitators can also provide translation services. The details of translation services are described in section 4.2.

Merging the messages can be useful for agents which are not designed to wait and handle several messages together. In our system, the facilitator provides this service so that all transactions a user agent has to handle is one-to-one single message transaction with the facilitator.

Sorting is sometimes a requirement for the system. User agents as well as facilitators can provide such a service.

Those functions which have been implemented for SAGE facilitators are delegation based on advertisement of category and agent capability, translation and merge of messages.

3 Implementation of the System and the Agent

In this section, we describe the implementation of the system of our inter-company EC application, SAGE:Francis, and the agents.

3.1 System Configuration

As mentioned in section 2, agents in the system are arranged like figure 1. This architecture is known as virtual catalog [3]. Virtual catalog has a 3-tier structure where facilitators are placed between user agents and database agents. User agents represent users as agents to SAGE, while database agents represent databases as agents to SAGE. Facilitators stand between them to provide brokerage, translation, merge and other services to help those agents interoperate.

We use only one facilitator for this current implementation, but plan to use multiple facilitators in the future. We are working on several methods to realize distributed facilitators.

Agents as programs has a configuration like figure 2. We use April for the transport level of agent communication [6]. There should be exactly one April communication server on each machine where agents are running. The April communication library in an agent process communicates with the April communication server to send and to receive messages between other agents which may be on the same machine or may be on the other machines.

The agent core with a KQML engine as its main component is what decides the agent's behavior. It is mainly driven by KQML messages the agent receives. But the agent core

[1] Those can be used effectively for the restriction of search range.

[2] Those are "who can access what."

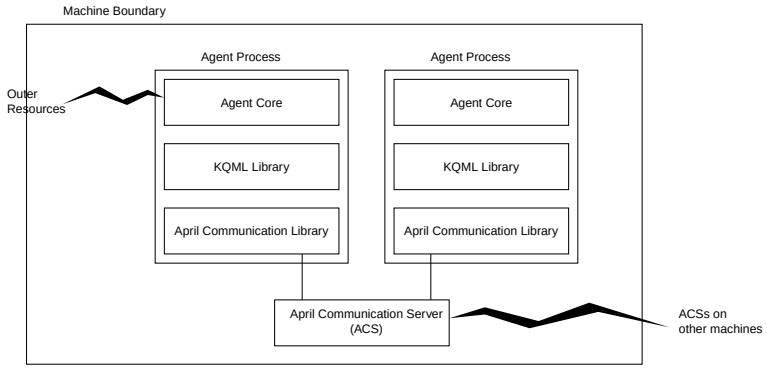

Figure 2: Agents as programs

might drive or be driven by outer resources. Such outer resources can be users or GUI (Graphical User Interface) for user agents and DBMSs for database agents. The agent cores are different from one agent to another and the details will be described for our agents in later sections.

Agents are supposed to have their virtual knowledge bases (VKBs) [5] in SAGE. A VKB is a set of KIF sentences in SAGE and transactions between agents are realized as accesses to each other agent's VKB.

KQML performatives implemented currently for SAGE:Francis are "`ask-all`," "`ask-one`," "`reply`," "`advertise`" and "`sorry`."

3.2 User Agents

A user agent process is configured with a CGI program at a web server, Java applets as in figure 3. The Java applet and the agent process are written in Java language, while the CGI program is written in perl.

A Java applet is downloaded into a browser and provides a user interface to SAGE (see figure 8). This applet sends the HTTP/POST message to the CGI program at a web server, which in turn relays the message to the agent process of the user agent and relays back the message from the agent process to the applet. The user agent process can handle messages from multiple applets simultaneously using threads.

There are several internal components in the user agent process. They are user management, KIF message manager (including KIF generator/parser), KQML engine, KQML library, April communication library.

In order to make user agents general-purpose and domain-independent, a Java applet and KIF generator/parser in the user agent process are made to be configured with a configuration file.

3.3 Facilitators

A Facilitator is written in Allegro Common LISP 4.3.1 and has an internal structure like figure 4. We chose LISP language because facilitators have to handle messages in S-expression and manipulate them in complex ways to provide useful functions.

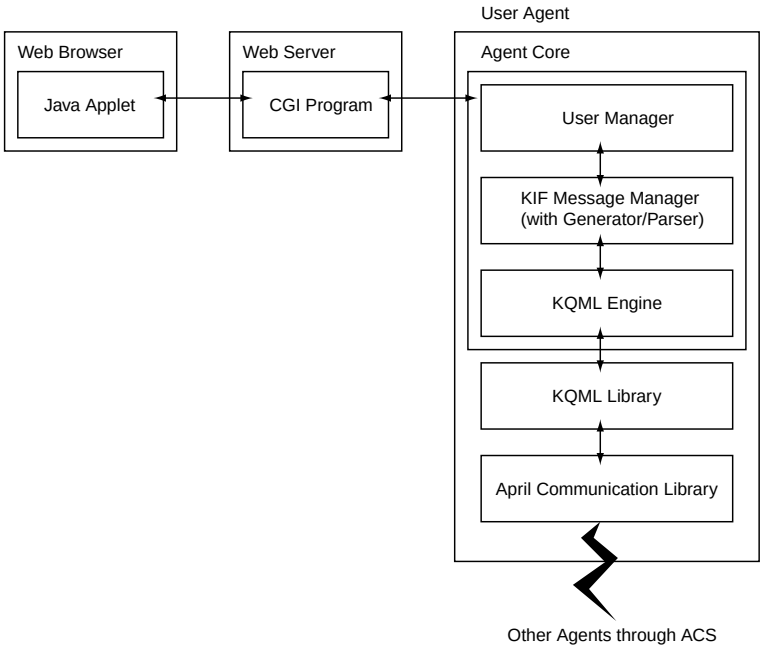

Figure 3: User Agent

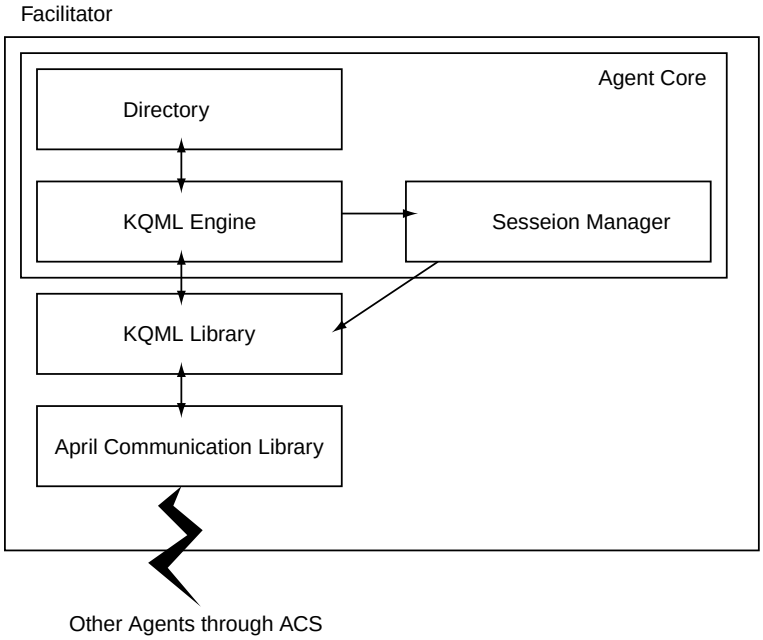

Figure 4: Facilitator

When advertise messages are sent to a facilitator, the directory stores the information on other agents such as categories and what kind of messages other agents handle and translation information. Currently the directory consists of lists with access methods. We have a plan to replace the directory with a Prolog-like inference engine.

The session manager keeps track of the message transactions. This module is what makes facilitators more than reactive agents. A "session" managed here is a series of related message transactions. Especially this module makes it possible for a facilitator to wait for multiple messages from database agents and then to merge them into one message to send back to the user agent, who sent the original message.

3.4 Database Agents

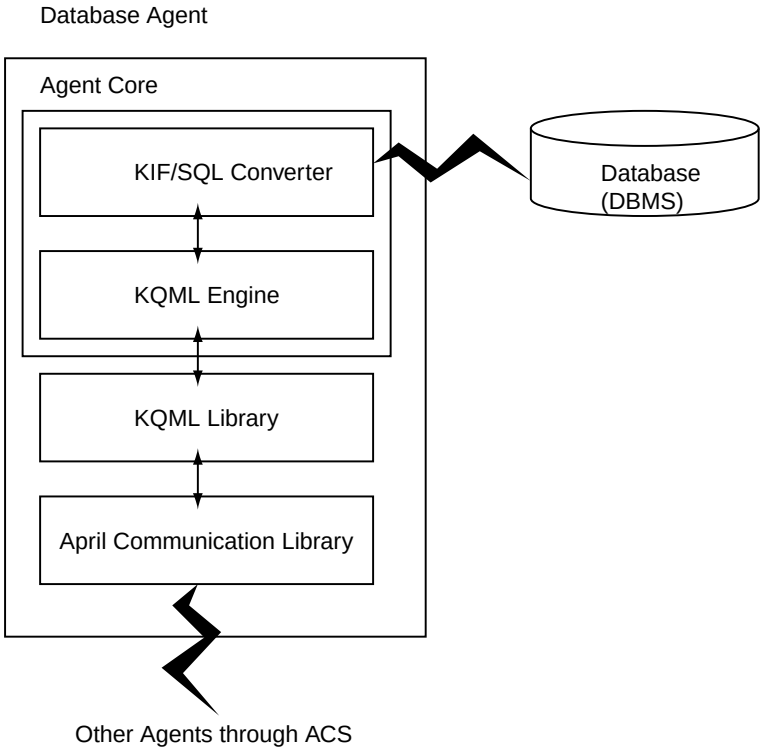

Figure 5: Database Agent

The structure of a database agent is like figure 5. When a database agent starts up, it advertises to the facilitator categories, message formats and ontology it can handle. After that, the main function of database agents is conversion between ACL messages (mainly KIF part) for other agents and SQL messages for the database. Database agents are written in Java and realized as wrappers to database management systems (DBMSs). With JDBC/ODBC bridge, we have successfully agentified Oracle and Microsoft Access DBMSs. For the case of Oracle, the database itself can be on a machine other than the one on which the database agent is.

4 Internal Workings

In this section, we describe how the system works with emphasis on the message flow and translation.

4.1 Message Flow

The message flow of the system is described in figure 6.

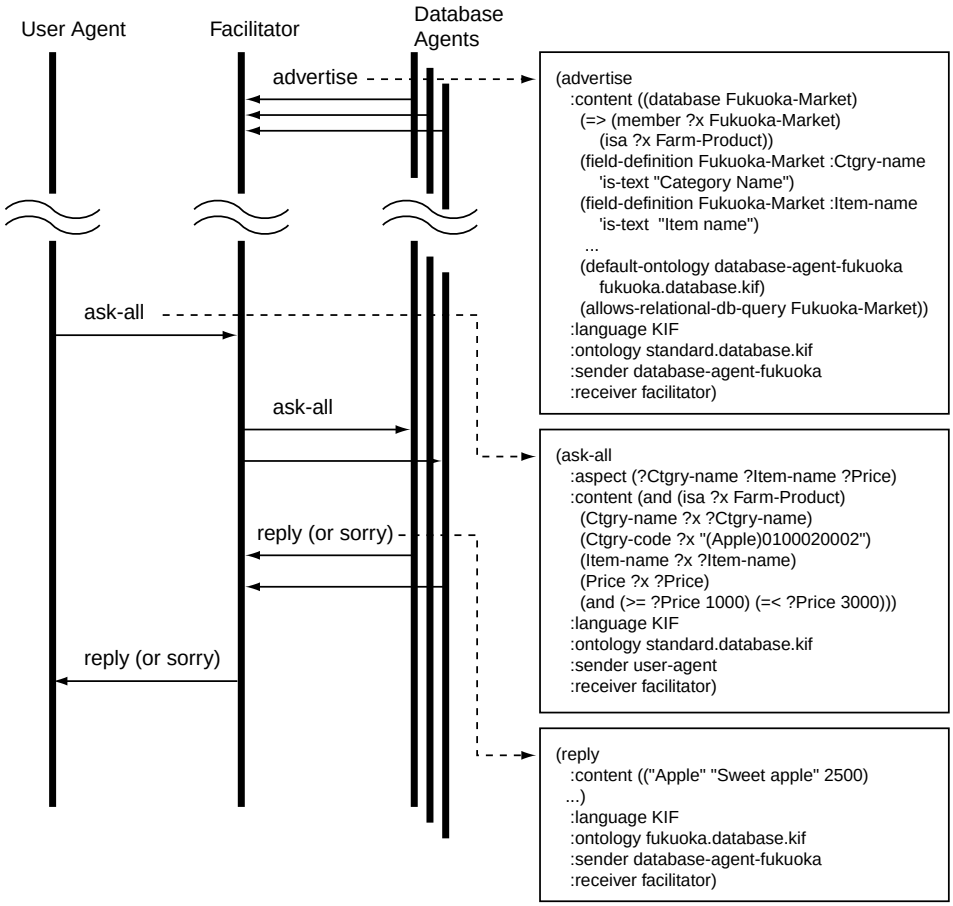

Figure 6: Message Flow: Messages in the figure are abridged for ease of understanding and display.

When a database agent starts, it advertises categories, message formats and ontology it can handle to the facilitator. Such information is stored in the directory of the facilitator (see figure 4).

A user makes a query through the GUI provided by the Java applet (see figure 8). Then the user agent turns the query into an ACL message and sends it to the facilitator.

The facilitator chooses the appropriate database agents that can handle the message and translates it for each database agent if necessary. Then the facilitator sends out the messages to the chosen database agents and wait for the reply messages.

The database agent, which receives the message from the facilitator, changes the ACL message into a SQL query and make a query to the database. The result from the

database is composed into an ACL message and the ACL message is sent back to the facilitator.

The facilitator merges the messages from the database agents into one and sends back to the user agent which made the original query.

Then the result is relayed to the Java applet, which displays the result in a list (see figure 9) and the details of each item (see figure 10).

4.2 Translation

Organizations joining inter-company EC projects usually have their own databases developed with their own terminologies. In order to make the integration of such disperse databases a reality, SAGE provides translation service for database agents, which is one of the main services facilitators provide.

Database agents advertise to the facilitators the names of ontologies they use. They can also advertise translation information as in figure 7. Or facilitators can load translation information from files. Facilitators then translate the contents of ACL messages based on the knowledge before sending the messages to the database agents.

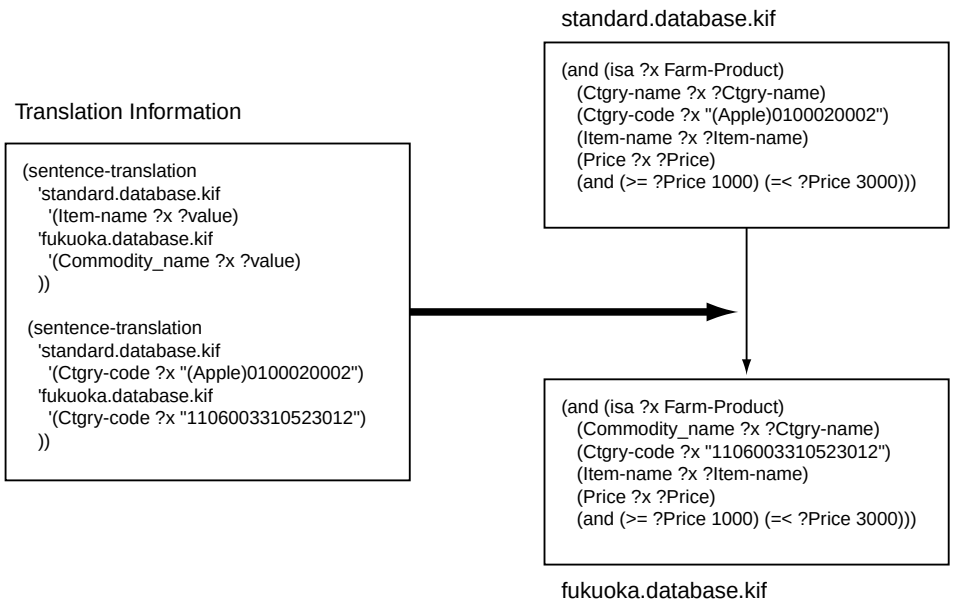

Figure 7: Translation

Facilitators' translation is done by searching a template KIF sentence through the original content and then, if found, replacing it to another matching KIF sentence. Template KIF sentences can have variables which can match any structure. The variables in the matching KIF sentence will be replaced with the matching structures before the matching KIF sentence replaces the template KIF sentence. With this mechanism, both predicates and values in KIF sentences can be translated.

Facilitators being ones who provide translation service, there must be someone who provides the content of translation from one ontology to another. That someone has to be a human-being who manages the database. Such a person is often an end user of computers.

Taking such situation into consideration, we plan to provide a visual tool, which can be used to align two ontologies and then to produce translation information for those two ontologies. We name the tool OAT (Ontology Alignment Tool) and working on it currently.

5 Experiments and Performance

We have set up the agent system for inter-company EC and carried out proof-of-concept experiments. We have also measured the performance of the system. The experiments and the measured performance are described in this section.

For the experiments, we wanted to use real data from real databases in use, but unfortunately it was not available for several reasons. Therefore we produced two databases, "Fukuoka Market" and "Kawasaki Market." They use different terminologies for field names and category structure for their commodities.

	Software/Language	OS	Hardware
Web browser	Netscape 3.01	Windows95	PC
Java applet	Java 1.0.2	Windows95	PC
Web server	Apache 1.1.1	Solaris 2.X	Sun Workstation
CGI program	perl 5.004	Solaris 2.X	Sun Workstation
User agent	Java 1.1	Solaris 2.X	Sun Workstation
Facilitator	Allegro Common LISP 4.3.1	Solaris 2.X	Sun Workstation
Database agent (Fukuoka)	Java 1.1	Windows95	PC
Database (Fukuoka)	Oracle 7.3	WindowsNT	PC
Database agent (Kawasaki)	Java 1.1	Windows95	PC
Database (Kawasaki)	MS Access95, 97	Windows95	PC

Table 1: Experiment setup: Sun workstations are Sun4/20's. PCs are Pentium 200 MHz machines. Of those above, there are groups of the processes, of which all the members run on the same machine. The groups are (1) the Web browser and the Java applet, (2) the Web server, CGI program and user agent, (3) database agent (Kawasaki) and database (Kawasaki).

The experiment setup is described in table 1. We used one Web server, one user agent, one facilitator and two database agents. Being set up this way, search requirements were put into a Java applet in a browser and a query was made. (see figure 8). Then the result

from the two databases was returned as a list (see figure 9) and the details of each item were displayed by clicking the button in the list. (see figure 10). The system worked out successfully.

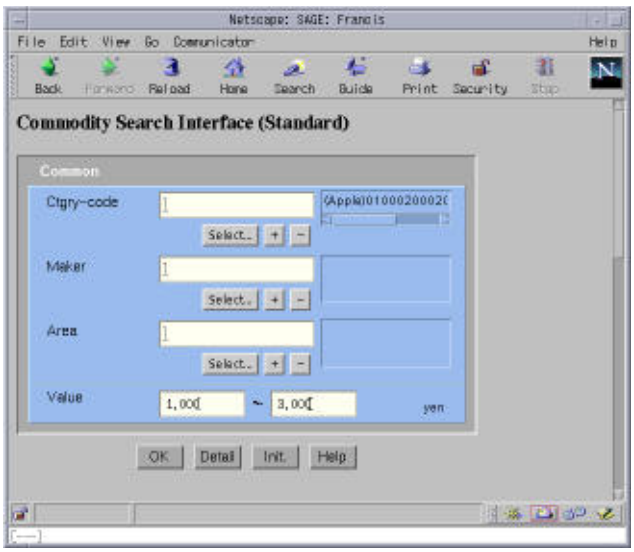

Figure 8: User Interface (1): Users enter search requirements into a Java applet.

Figure 9: User Interface (2): The result of the search is returned as a list.

Experiments are also done for a Japanese version as well as the English version reported in this paper.

High performance of the system is a very important requirement, as we are going to apply the system to real-world problems. Even a system with useful functions is worthless if its performance is poor.

With the same setup as above, performances are measured for two values, (1) the response time and (2) the throughput rate. The response time is the time between when the user clicks the "O.K." button to send a query message and when the Java applet starts to display the result list. The second throughput rate is measured in terms of CPU time consumed in each agent. Current requirements for those values are 10 seconds for (1) and 5,000 queries an hour at a facilitator for (2).

The response time was measured and averaged for 10 trials. The average was 7.1 seconds. The response time for a query should be the sum of the time elapsed on the

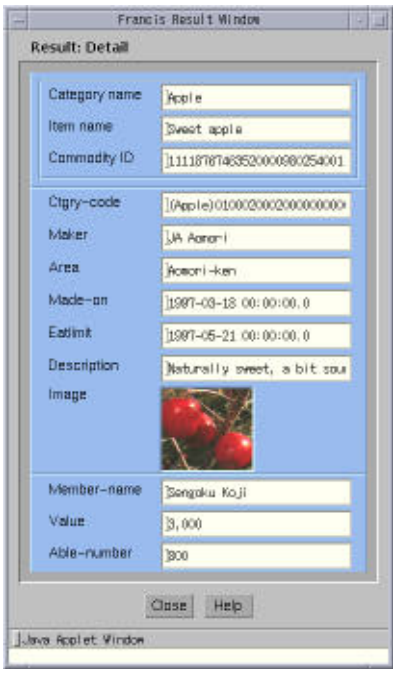

Figure 10: User Interface (3): The details of each item is displayed by clicking the button in the list.

each part of the route of the messages. [3] The table 2 gives the elapsed time in each agent process. They do not sum up to 7.1 seconds since there is time elapsed in Web server and in communication, which we could not measure.

Theoretical throughput can be calculated from the CPU time consumed in an agent. The table 2 gives average CPU time over 10 trials to handle a query for a facilitator. The CPU time for facilitator includes receiving the message from user agents, choosing, translating, sending the message to two database agents, receiving messages from the database agents, merging the messages and sending the message back to the user agent.

5,000 queries an hour allows a facilitator to use around 0.72 seconds for each query. 0.3 seconds is allowable in that respect.

Since CPU time for one message may not represent real load when many messages are sent to the facilitator, we plan to make a sandbag-like system to load the facilitator with many messages to check the performance in a more realistic situation.

6 Summary

We have reported, in this paper, SAGE and its application to inter-company EC, implementations of the agents, how the system works, experiments and its performance with emphasis on functions of facilitators. The system is actually built and proof-of-concept experiments have been carried out. Performance measurement showed us that the system is within real-world requirements.

Other application areas in which we are currently working on, include knowledge management in enterprises and integration of online databases. They are also close to real

[3] To be exact, the response time for a query is the maximum of the sum over different paths. Messages can take different paths for one query.

	Elapsed Time	CPU Time
User agent	0.07 + 1.54	
Facilitator	0.18 + 0.16	0.3
Database agent	2.32	$\sim$ 0.7

Table 2: Elapsed Time and CPU Time in seconds: Elapsed time in agent processes is measured for all agents. As for elapsed time for the user agent and the facilitator, the first number is for query messages and the second number is for reply messages. Elapsed time for database agent does include time elapsed in the DBMS. The number 2.32 is of Fukuoka database agent (Oracle), which was longer than 1.58 of Kawasaki database agent (MS Access). The CPU time for the facilitator is measured using a LISP profiler. Since Java profilers were not available, exact CPU time was not measured for the user agent and database agent. The CPU time for the database agent was estimated to be less than 0.7 seconds by the fact that it handled 10 messages in 7 seconds.

deployments.

Future work includes realizing faster implementations of agents, implementing other useful functions of facilitators, and distributed implementation of facilitators.

References

[1] M.R.Genesereth and S.P.Ketchpel, "Software Agents," Comm.ACM Vol.37 No.7, 1994.

[2] "Conversational Agent," IEEE Internet Computing Vol.1 No.2 pp.73-75.

[3] A.M.Keller and M.R.Genesereth, "Multivendor Catalogs: Smart Catalogs and Virtual Catalogs," 1996.

[4] M.R.Genesereth and R.E.Fikes, "Knowledge Interchange Format Version 3.0 Reference Manual," Technical Report Logic-92-1, Computer Science Department, Stanford Univ., 1992/6, URL: http://logic.stanford.edu/papers/kif.ps

[5] The DARPA Knowledge Sharing Initiative External Interfaces Working Group, "Specification of the KQML Agent Communication Language," 1994/2/9, URL: http://logic.stanford.edu/papers/kqml.ps

[6] McCabe, F. G. and Clark, K. L.: "April - Agent PRocess Interaction Language," Lecture Notes in Artificial Intelligence 890, pp. 324-340, Springer-Verlag, 1995

[7] URL: http://infomaster.stanford.edu/

[8] URL: http://www.fujitsu.co.jp/hypertext/solution/industry/Package/Witweb/wit.html

[9] URL: http://www.hitachi.co.jp/Prod/comp/ec/twx21/index.html

[10] URL: http://www.tradeex.com/