SAGE and Its Application to Electronic Commerce

- SAGE:Francis, A System Based on Virtual Catalog -

Tamami SUGASAKA, Ryusuke MASUOKA, Akira SATO,
Hironobu KITAJIMA and Fumihiro MARUYAMA
Fujitsu Laboratories Ltd.
2-2-1 Momochihama, Sawara-ku, Fukuoka 814, Japan
{tamami, masuoka, satoyan, kitajima, superb}@flab.fujitsu.co.jp

Abstract

Under the framework of the SAGE (Smart AGent Environment) project, we are investigating the seamless integration of information distributed over networks. In SAGE, conversational agents communicate over ACL (Agent Communication Language) and cooperate in solving a problem. To put SAGE into practical use, we are presently trying to employ this system in electronic commerce.

We have constructed a first prototype called SAGE:Francis. The prototype has a search function that is based on compound terms. We had already collected results from an experiment with two legacy database applications, Oracle and Access, who used differing names and category structures. SAGE:Francis successfully integrated information on commodities that was stored in these databases and we achieved an average response time of 5 seconds.

In this paper, we propose a new model of SAGE:Francis that provides this function and discuss the ontology it establishes for EC.

1 Introduction

In recent years, network infrastructures spread rapidly and the informational society has made continuous progress. Therefore, a technology is required that is able to support a seamlessly integration of necessary information from various information sources. We put the emphasis of our work on an integration of different information sources, a reuse of existing resources (mainly databases), and the implementation of a practical system.

In order to integrate and access sources of information which are distributed over the network and use differing formats depending on the respective provider, and to ensure furthermore a smooth cooperation with respect to this information, the following features are required: A common communication protocol between provider side and user side, a mechanism for selecting the most suitable source for requested information, a translation function for differing terms between user side and provider side, and also a translation function for irregular terms between providers.

We attempt to solve the above mentioned problems by employing software agents that communicate in ACL (Agent Communication Language). With such agents in mind, we are developing a smart agent environment (called SAGE [11]). The main research issues in connection with the development of SAGE are as follows: (1) Creating agent as representation of users who access information through user interfaces and of information sources such as offered by legacy database management systems (DBMSs), (2) Constructing a mediator agent (called facilitator) that would forward a query to suitable agents and translate terms that differ among the agents, (3) Specifying an agents communication language (ACL) for describing formats and activities in connection with message exchange.

In order to test the applicability of SAGE to a real-world system, we tried to apply this system in Electronic Commerce (EC). This test application was called SAGE:Francis.

The area of EC can be divided into two categories: Consumer EC and inter-company EC.

Consumer EC covers such areas as electronic shopping on the Internet by providing an electronic mall. The general approach for achieving this is as follows: Dealers that are represented on the Internet gather under a single address and establish on electronic mall. In this case, a reference menu for the site is prepared in advance and information about the goods the dealers offer are virtually put together into a single list display that follows certain pre-selected conditions. As commodity information is shown in an index that is linked to the individual sites, the available goods of the electronic dealer are seamlessly displayed in the electronic mall and comparisons become easy. Within the area of consumer EC, the points to be clarified are how to build a suitable index server, how to collect commodity information, and how to register commodity information.

On the other hand, the area of inter-company EC has to support a variety of differing functions such as search for products, price negotiations, procurement and payment. Several enabling technologies for supporting these functions are available in the market, including WitWeb from Fujitsu, TWX-21 from Hitachi and TRADE'ex from TRADE'ex. These systems use a centralized server that provides the related services. However, as the size of the system grows, it becomes necessary to support the use of distributed servers. It might also be required to integrate these systems.

We try to employ SAGE, which integrates two or more legacy databases with agents, in the area of server cooperation, which is the area of the above-mentioned issues that occur in connection with inter-company EC. Our goal

is to cover the entire purchasing process, such as searching for commodity information that is distributed over several sources, narrowing down the decision by comparing the offers and finally negotiating the purchase conditions.

2 SAGE:Francis

2.1 Virtual Catalog

SAGE:Francis is based on the "Virtual Catalog" technology [3] that was proposed by Stanford University.

This technology was meant to solve the following problems in connection with online catalogs:

- It is difficult to search the databases according to a pre-defined purpose.
- A lot of navigation (through menus etc.) is required to search according to a pre-defined purpose.
- The structures, terms etc. that are used in the catalogs of the respective companies differ from each other.
- It is difficult to search the database according to contents.
- It is difficult to search through two of more catalogs.

When using Virtual Catalog, catalogs that are suitable with respect to a specific request from a user are automatically selected from among the separate catalogs of the various companies. The separate catalogs might use different formats and terms, but they are shown as a single, virtual catalog that integrates all suitable catalogs. The agents of users and sources communicate via Agent Communication Language (ACL), and a mediator agent ("facilitator") moderates between these agents by such functions as selecting suitable agents, and translating terms. The facilitator is also one of the agents and communicates via ACL.

As a sample implementation of Virtual Catalog, Stanford University has developed the "Infomaster"-application, which provides housing information from newspapers, campus-wide job information etc. However, Infomaster is a system that was merely set up for verification purposes. In order to make it practically usable, several issues require further examination.

One of these issues is a lack of adaptability with respect to agent extension, as user agents and facilitators form one unit. A decentralization of agents by mirroring etc. is expected as the system becomes larger, and this will make it necessary to be able to easily make extensions to the agents. As long as the agents are not separate, adding more extensions for improving and increasing the functions of agents requires on a larger scale.

Another issue is the inability of updating data in real time, as processed data has first to be incorporated into the agent at the provider side.

To address the above mentioned issues for in connection with the utilization of the virtual catalog, we provided the following features during the development of the prototype SAGE:Francis.

- Agents are implemented independently from each other.
- Data is managed by legacy DB applications.

2.2 Functions and Architecture

In the current version of SAGE:Francis, a search function for inter-company EC, was implemented.

Commodity search by compound condition is a function that allows to specify compound conditions and transactions for the commodities, such as categories, places of production, producers, prices, quantities and etc.

List display and detailed display is a function that supports the comparison of commodities and displays more detailed information.

The architecture of SAGE:Francis is shown in figure 1.

SAGE:Francis consists of user agents (UA), facilitators (FA) and database agents (DBA). All agents communicate via ACL. One of the main functions of UAs is to make a conversion between ACL and the Internet-browser language HTML. One of the main functions of the DBA is to make a conversion between ACL and SQL, the language of such DB applications, as Oracle or MSAccess. Other key functions include converting DB data into virtual knowledge that can be used by a DBA, and to inform FAs of the capabilities of DBAs (a function that is called "advertising"). The main functions of the FAs, which act as mediators between UAs and DBAs, consist of forwarding message to suitable agents and integrate results, replying accordingly if there are no suitable agents, and translating terms between agents.

On the background of the structure of SAGE:Francis, it is necessary to apply following techniques: (1) abstracting from the sources of information (DB) and the interface between agents, (2) representing users and DBs by agents, (3) constructing FAs and translating term.

With respect to (1), it is important to determine how the commodity information an agent has should be expressed. It is also important to specify ACL in such a way that search by compound condition and displaying a list of results become possible. Moreover, the elements of the "advertise" function have to be determined. With respect to (2), it

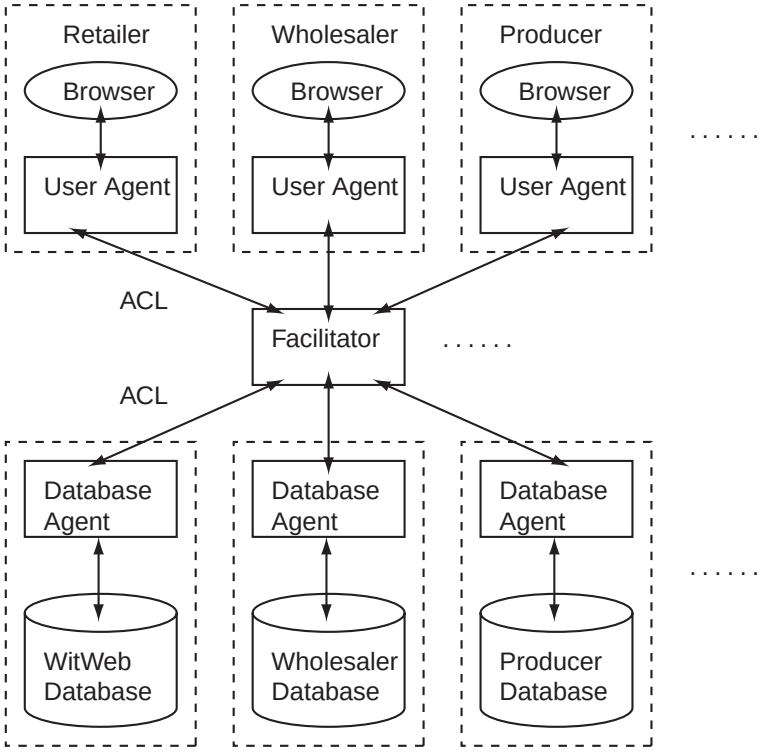

Figure 1: Architecture of SAGE:Francis

is important to create a representation of users and DBs by agents in such a way that it becomes easy to customize the search conditions and the specifications of the requested information. It is also important to determine ways for communicating with legacy DB applications (SQLDB) via SQL. With respect to (3), it is important to forward messages by category, and to translate terms between the ontologies that are the term systems of the various providers.

3 Applied Techniques and Implementation

3.1 VKB (virtual knowledge base)

To enable agents to communicate via ACL, It is necessary to abstract from the available information to the knowledge (virtual knowledge base:VKB) that is available to an agent. In SAGE:Francis, the original information is stored in DBs, and the abstracted information is collected in the VKB of the DBA.

The VKB that was provided for the search function of SAGE:Francis was defined as "a subset of the record in which an operation to reference a specified field value is possible". The definition of VKB by KIF [4] is as follows:

```
(defrelation database (?db) :=
  (and (set ?db)
       (exists (?set-of-fields)
               (and
                 (set ?set-of-fields)
                 (not (empty ?set-of-fields))
                 (forall (?field)
                         (=> (member ?field ?set-of-fields)
                             (arity ?field 1)))
                 (forall (?record ?field)
                         (=> (member ?record ?db)
                             (member ?field ?set-of-fields)
                             (defined (value ?field ?record))))
))))
```

"?db" is a set, and "?set-of-fields" is a subset of "?db". For arbitrary elements "?records" of "?db", and arbitrary elements "?fields" of "?set-of-fields", the function "(field-value ?record ?field)" is defined. "(field-value ?record ?field)" is a function which returns the value of "?field" specified via "?record".

3.2 Interface between the agents

Information from the VKB is used by extracting knowledge that is described by certain conditions and use it according to specifications. When accessing VKB via ACL, knowledge is extracted from the VKB according to KIF statements. The operations that are to be applied to that knowledge are specified by commands (called "performative" of KQML [5]).

This approach leads to the following implementation of the search operation in SAGE:Francis.

3.2.1 Performative of KQML

In the current implementation of SAGE:Francis, four kinds of KQML performatives are used:

ask-all/ask-one Asking a question.

reply Replying with a result.

advertise Announcing capabilities.

sorry Answering that "reply" is impossible.

3.2.2 Relation of KIF

In the current implementation of SAGE:Francis, four kinds of relationships are supported:

1. Matching of a specified field with its value in a VKB record (commodity name, category, etc.)
2. Arithmetic comparisons of numerical values (=<, >=, etc.)
3. Logical combinations (and, or, etc.)
4. Secondary information (such as number of records, total sum, average, maximum, minimum, first element, last element, etc.)

3.3 "Advertise" Function

DBAs inform FAs of their capabilities so that efficient access from other agents becomes possible. In the current implementation of SAGE:Francis, the following elements of information about a VKB are announced by the "advertise" function.

- The name of the VKB
- The categories that is covered by the VKB
- Fields of the VKB
- The ontology that is used for the VKB
- Relation that can be used for accessing the VKB

An example for the "advertise" function is shown in figure 2.

```
((database FukuokaMarket)
 (=> (member ?x FukuokaMarket)
  (isa ?x agricultual-products)
  (field-definition FukuokaMarket commodity-name 'is-text)
  (field-definition FukuokaMarket category-name 'is-text)
  (field-definition FukuokaMarket category-code 'is-number)
  (field-definition FukuokaMarket producer 'is-text)
   ...
  (default-ontology standard.database.kif)
  (allows-relational-db-query FukuokaMarket)))
```

Figure 2: Example of advertise

In this example, the DBA "advertises" the following information: the name of the VKB is "FukuokaMarket", the treated category is "agricultural products". The fields in the VKB are "commodity name", "category name", "category code", "producer" etc., and the ontology is "standard ontology". Moreover, KIF relation which can be used for access, such as

(allows-relational-db-query FukuokaMarket)

are advertised (or announced).

3.4 User Agents (UAs)

The main function of UAs is to translate search requests from the WWW browser into ACL messages. The architecture of these UAs is shown in figure 3.

The UA consists of a display system on the user side and a process system on a server side. The server side and the user side communicate via HTTP. After receiving a message from the user side, the httpd function on the server side communicates with the process system through a CGI or a NSAPI.

GUI A search condition and its result are displayed by an applet. Moreover, in order to be able to efficiently customize the GUI according to the respective application field, the applet is generated from an input-and-output format file.

HTTP communication module The server side and the user side communicate over the same encoding method (URL-encoded keyword-value pairs) which is used for HTML formed.

User management module This module manages user information so as to provide user authentication for an applet and for generating messages to a user.

KIF generation module This module generates KIF messages according to user requests.

KIF message management module This module stores the contents of KIF message and analyzes for creating an appropriate reply message.

KIF analysis module This module analyzes KIF reply messages based on the original message, and generates suitable matches of items and values.

KQML engine module This module generates KQML messages in the module of KQML generation. Moreover this module analyzes KQML messages and determines suitable functions in the module for KQML analysis.

Agent communication module Through this module, UAs communicate with other agents via the April communication library. This module stores all received messages which may be read from other modules via function calls.

3.5 Database Agents (DBAs)

One of the main function of DBAs is translation between ACL and SQL, where SQL is a language of DB applications (Oracle, Access). Another key function is advertising the available knowledge and capabilities of the DBA to FAs. The architecture of DBAs is shown in figure 4.

Agent communication module This module is identical to the agent communication module of the UA as described in Section 3.4.

KQML engine module This module controls the timing of the "advertise" function in a module for advertise control, analyzes KQML messages in a module for KQML analysis, and generates KQML messages in a module for KQML generation.

KIF $\leftrightarrow$ SQL converter module This module converts KIF messages into SQL commands and accesses DBs through ODBC or JDBC. Moreover this module converts the messages from the DB into KIF messages.

Database Data is managed with legacy DB applications, such as Oracle and MS Access.

3.6 Facilitator (FA)

The architecture of the FA is shown in figure 5.

Agent communication module This module is identical to the agent communication module of the UA as described in Section 3.4.

KQML engine module In a module of KQML analysis, KQML messages are analyzed and the included performative processing modules are called. The messages initiate processes with information that is stocked in the directory maintenance module of each processing module, and the result is sent to the Session Management module.

Directory maintenance module This module consists of a module for the correspondence to DBA categories, a module for the correspondence to DBA messages, and a translater module. It has the functions of storing "advertise" information (such as categories and accessible fields) from the DBA, and to execute requests performative processing modules in the module for KQML analysis. The module for the correspondence to DBA categories replies with DBAs that can use the category. The module for the correspondence to DBA messages replies with DBAs that can use the message. The translator module translates messages into term that the DBA can use.

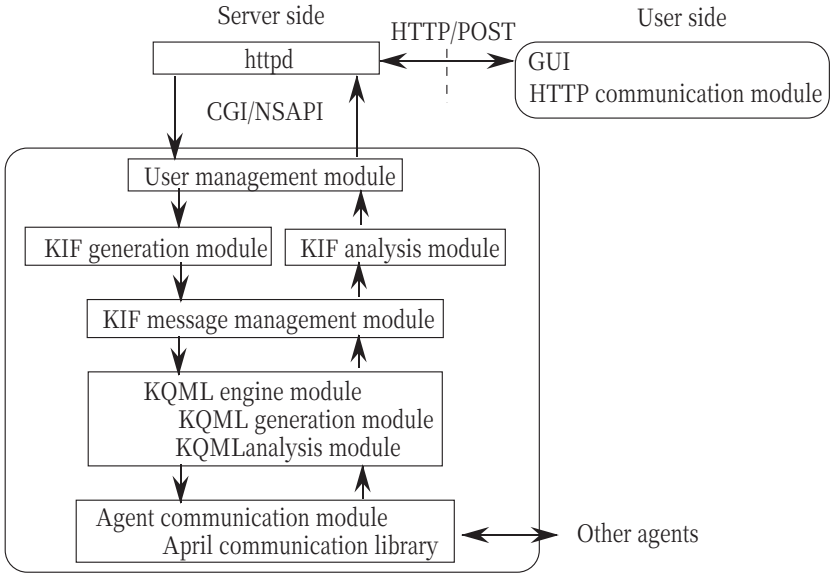

Figure 3: User Agent

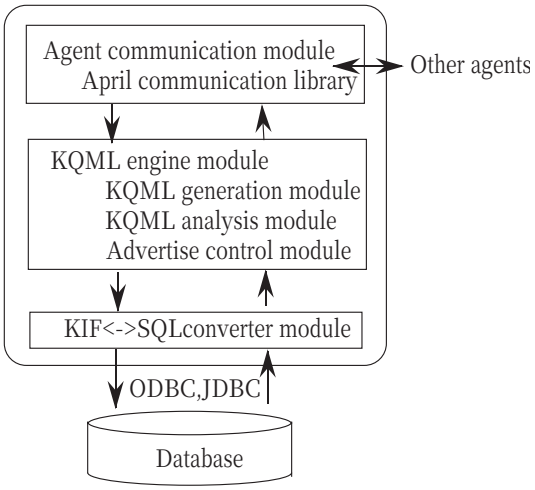

Figure 4: Database Agent

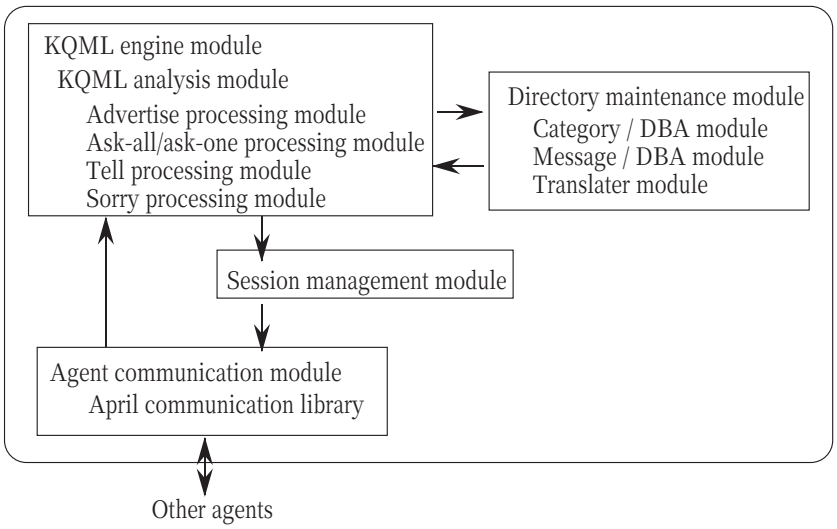

Figure 5: Facilitator

Session Management Module This module manages message exchange. In the current phase of the implementation, a session is defined as the activity from sending out a query from an UA to DBAs to sending out the replies from the DBAs to UAs. In this module, it is confirmed whether a timeout occurred and whether the conditions for returning a reply were satisfied for every fixed time. In the case of a timeout, a message containing the results that were collected until that time is sent out through the Agent Communication Module. In the case of a success, a message containing all results that were collected is sent out. In all other cases, a "database agents are not available" message is sent out.

3.7 Translation

Organizations that take part in inter-company EC projects usually have developed their own databases which have their own terminologies. In order to make the integration of these separated databases a reality, SAGE provides a message translation service for database agents, which is one of the main services that the facilitators provide.

Database agents advertise to the facilitators the names of the ontologies they use. They can also advertise information related to message translation as in figure 6. Alternatively facilitators can load translation related information from files separately. Facilitators then translate the contents of ACL messages based on this knowledge before sending the messages to the database agents.

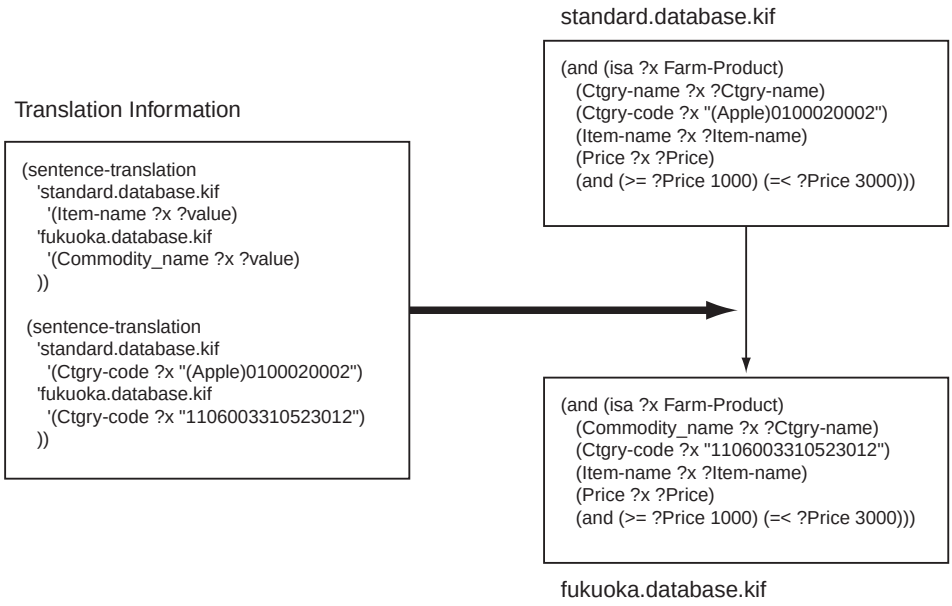

Figure 6: Translation

Message translation is achieved by searching for a template KIF statement within the contents of the message and replacing it by an appropriate KIF statement, if possible. Template KIF statements may contain variables to match various structures. The variables in the matched KIF statement will be replaced with the appropriate elements before the template KIF statement is replaced. Through this mechanism, both as well as values in KIF statements can be translated.

While facilitators provide translation services, there must nevertheless be an instance that determines the actual contents of translations from one ontology to another. This function has to be performed by the person who manages the database.

For this reason, we plan to provide a visual tool for aligning two ontologies and subsequently producing translation information between those two ontologies. We gave this tool the name OAT (Ontology Alignment Tool) and are currently working on its development.

3.8 Message Flow

The message flow of the system is described in figure 7.

When a database agent starts, it forwards a advertising message of its categories, message formats and ontology it can handle to the facilitator. This kind of information is stored in the directory of the facilitator (see figure 5).

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

The facilitator chooses the appropriate database agents for handling the message and translates it for these database agents, if necessary. The facilitator then sends out the messages to the chosen database agents and waits for the reply messages.

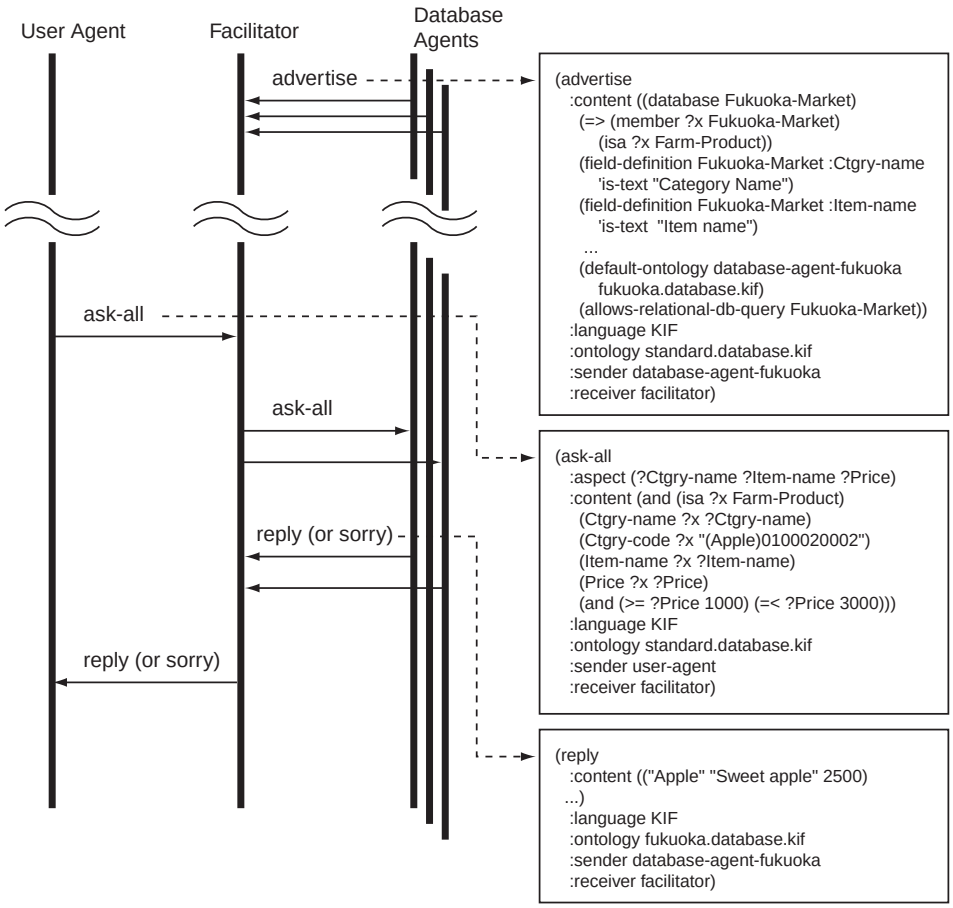

Figure 7: Message Flow: Messages in the figure are abridged for ease of understanding and display.

The database agent which receives the message from the facilitator changes the ACL message into an SQL query and performs a database query. The result from the database is encoded into an ACL message which is sent back to the facilitator.

The facilitator merges the messages from the database agents into a single one and sends it back to the user agent which issued the query originally.

The result is sent to the Java applet, which displays a list of the results (see figure 9) as well as the detailed information of each item (see figure 10).

4 System Tests and Performance Results

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

We originally wanted to use the data from real-world databases in our tests, but found unfortunately that such data was not available for various reasons. Therefore we set up two new databases, "Fukuoka Market" and "Kawasaki Market." These databases use different field names and category structures for the commodity data they stored.

The setup of the test is described in table 1. We used one Web server, one user agent, one facilitator and two database agents. Search requirements were then inputed into a Java applet via a browser and a query was produced. (see figure 8). The results from the two databases were returned as a list (see figure 9), and it was possible to see the detail information by clicking the appropriate button in the list. (see figure 10). The system worked successfully.

Tests were performed for a Japanese version as well as for the English version described in this paper.

High performance of the system is a vital requirement, as we plan to apply the system in real-world projects. Even a system that provides useful functions might be worthless if its performance turns out to be too poor.

For the same setup as described above, we measured the performance with respect to two parameters: (1) response time and (2) 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 list of results. The throughput rate was measured in terms of the CPU time used by each agent. The requirements with respect to these values at this stage are 10 seconds for (1) and 5,000 queries an hour per facilitator for (2).

The response time was measured and averaged for 10 trials: We obtained an average of 7.1 seconds. It should be possible to calculate the response time for a query by adding up the elapsed times in each part of the route of the

	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).

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.

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

messages. [1] Table 2 shows the elapsed times for each agent process.

The theoretical throughput can be calculated from the CPU time used by the agents. Table 2 shows the average CPU time over 10 trials that elapsed when handling a query at a facilitator. The CPU time for the facilitator included 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.

At a rate of 5,000 queries an hour, the facilitator can use about 0.72 seconds for each query. 0.3 seconds is a good result in that respect.

Since the CPU time for one message may not represent the actual work load when many messages are sent to the facilitator, we plan to setup a buffer system for loading the facilitator with many messages at once, so as to check the performance in a more realistic situation.

5 Conclusion

We introduced in this paper the SAGE system and its application to inter-company EC. We also discussed the implementation of the agents, how the system works, tests we performed and the performance of the system with emphasis on the performance of the facilitators. The system was actually implemented and proof-of-concept tests were carried out. Performance measurements showed that the achieved system performance is sufficient for real-world requirements.

Other application areas we are currently working in, include knowledge management in enterprises and integration of online databases. These projects also have close relationships real-world deployments.

[1] To be exact, the response time for a query is calculated as the maximum of the response times for different paths, Since messages for a query can take different paths.

	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.

Our future work will includes implementations of faster agents, implementing more useful functions of facilitators, and implementation of distributed 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/

[11] R. Masuoka, T. Sugasaka, A. Sato, H. Kitajima and F. Maruyama, "SAGE and Its Application to Inter-company EC", Proceedings of PAAM98, pp.123 - 135.