UDC 002.66: 672: 681.3.02

Multi-agent System for Virtually Integrated Distributed Databases[note)]

●Yuji Takada ●Takao Mohri ●Hiroyuki Fujii

(Manuscript received October 22, 1998)

To achieve collaborations by sharing information in inter-enterprise environments such as CALS, a method of accessing seamlessly databases distributed over a network is required. In this paper, we show a multi-agent system which virtually integrates distributed databases. With this system, users can access seamlessly databases as if accessing a single database. As benefits of the multi-agent architecture, our system has features desirable for information sharing in inter-enterprise environments; each enterprise can set up independently the policy for maintenance, management, and security on the system. The experimental system was built up under the Steel Plant CALS project in Japan, which proved the efficiency and effectiveness of our method.

1. Introduction

One of the aims of CALS (Commerce At Light Speed) is to achieve collaborations by sharing information in inter-enterprise environments.[3)] Sharing information within several enterprises enables one enterprise to collaborate effectively with others and to achieve high productivity of collaborating enterprises. The Nippon CALS project extends the CITIS (Contractor Integrated Technical Information Service) model of the Department of Defense of the United States to inter-enterprise environments and proposes a bilateral CITIS model which allows both a demanding enterprise and a supplying one to share information in common.[1), 2)]

Note) A part of this work has been done in the Steel Plant CALS project sponsored by MITI (Ministry of International Trade and Industry, Japan) through "the Inter- and Intra-Corporate Electronic Commerce Promotion Projects" of IPA (Information-technology Promotion Agency, Japan). A preliminary version of this paper appeared in the "Proceedings of CALS Expo INTERNATIONAL 1997."

For this kind of information sharing in inter-enterprise environments, an important information technology is a method of accessing seamlessly databases distributed over a network, which are managed by different enterprises. In the Steel Plant CALS project,[13)] we investigate this method in order to share information on plants, for example, information on spare parts of various equipment held by steel-making companies and equipment manufacturers.[12)] With this, a steel-making company can quickly find spare parts to repair equipment such as rolling steel mills not only from within the company but also from equipment manufacturers and even plants of other steel-making companies. This quick finding is very important for steel-making companies, since a halt of rolling steel mills in longer time means larger loss of profits. Steel-making plants tend to be close together in Japan, for example three plants in the Tokyo bay area, but currently it is hard to find spare parts from neighboring plants.

One of the characteristics of systems that share information in inter-enterprise environ-

ments is that databases are necessarily distributed over a network because of the security issue of information and difficulties of sharing maintenance and management of databases. Enterprises are sensitive to provide others with their own information. Also, sharing databases means sharing maintenance and management within these enterprises. These situations make it not realistic to have a single database of all the data from all the enterprises. For a single database, enterprises would become more sensitive on their information and would not offer so much information. Also, the policy of maintenance and management of the database, including the cost distribution, would become a big issue. With distributed databases, users by themselves have to access several databases via a network. If the scale of collaboration is large and therefore the number of databases is also large such as in CALS, even finding databases suitable to requests becomes severely difficult for users.

In this paper, we show a system which enables users to access seamlessly to databases distributed over a network as if accessing to a single database. Since the distributed databases act as a single database from the view of users, we call the distributed databases enhanced with this system the *virtually integrated database*.

We build the system as a *multi-agent system*. An *agent* is a process that works by itself and also collaborates with other agents by message passing. A multi-agent system is a distributed system where one or more agents collaborate over a network. In our method, we assign an agent to each user and each database and arrange over a network agents which intermediate between other agents. For each access request from a user, the system finds databases suitable to the request by forwarding the request from agents to other appropriate agents according to conditions on the request and the information which the agents themselves manage. Then, user agents and database agents access to selected databases in parallel by their collaborations. Since users have only to access to this multi-agent system in order to access to distributed databases, the distributed databases work with the multi-agent system as a virtually integrated database.

Our system offers a way to solve the security issue and difficulties of sharing maintenance and management necessary to be solved in inter-enterprise environments. Each enterprise can set up independently the policy for security, maintenance, and management on the system. It also offers load balancing, local maintenance, scalability, and robustness. These are benefits of building the system as a multi-agent system.

The Steel Plant CALS project built an experimental system based on our idea and had experiments on collaborations by information sharing within steel-making companies and equipment manufacturers. These experiments showed that a virtually integrated database seemed to be useful to make the maintenance of equipment more effective and efficient. We also mention about implementation details and performance of this experimental system.

2. Logical configuration of virtually integrated database

To organize databases distributed over a network as a virtually integrated database, we shall set up a system called the *facilitator*[7] between users and databases as shown in **Figure 1**. The facilitator intermediates between users and databases in the following way:

- The facilitator has information about the databases and, based on this information, it selects databases suitable to a request from a user.
- The facilitator accesses the selected databases with the request and then collects and unifies the results from the accessed databases and replies to the user.

To access distributed databases, users have only to access the facilitator. The facilitator hides all the databases from users and pretends as a single database. Therefore, if we set up the facil-

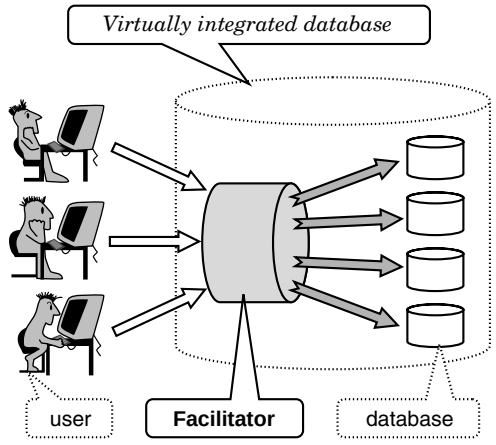

Figure 1
Logical configuration of virtually integrated database.

itator then the databases work as a virtually integrated database with the facilitator.

A straightforward realization of this logical configuration is to construct a single facilitator on a network, which manages all the databases and handles all the requests from users. When the system is used in a single enterprise and the scale of the system is relatively small, this realization is suitable, since the system is not so complicated and its maintenance and management can be focused on the facilitator. However, in the inter-enterprise environment where the scale of the system tends to become larger, this approach causes the following issues due to the centralized architecture of the facilitator:

Load: The facilitator must handle all the requests from users and all the replies from databases. Therefore, as accesses increase more and more, the load of the facilitator becomes higher and higher. As a result, the performance of the system becomes worse and worse. This over loading problem limits the scale of the system.

Maintenance: Registering new users and new databases to the system and withdrawing registered users and registered databases from the system cause modifications of the facilitator. Modifying user interfaces and database interfaces also cause modifications of the facilitator. These modifications tend to be harder if the scale of the system is very large. Moreover, when these modifications require the facilitator to halt, the total system itself halts even if distributed databases are working. These mean that the maintenance of the system is hard.

Share: When several enterprises share a single facilitator, it must be decided which enterprise manages and maintains the facilitator. This implies that the cost for maintenance and management must be distributed. As more companies share the facilitator, the way of distribution becomes a harder problem. Moreover, the policy of the managing enterprise may dominate others on the maintenance and management of the facilitator. Then other enterprises may have to follow the policy, which may include the cost distribution.

Security: More information about the databases the facilitator has, better it can select databases suitable to requests. Therefore, the facilitator should have as much information about databases as possible. However, the information may include some secure one and, if the facilitator were managed by one enterprise, others would be afraid that this secure information could not be protected. Hence, security issue limits the information about databases for the facilitator.

Especially, *share* and *security* are critical issues in inter-enterprise environments. These issues imply that the straightforward realization with one centralized facilitator is not realistic in inter-enterprise environments such as CALS.

3. Multi-agent architecture of facilitator

An *agent* is a process that works by itself and works better by collaborating with other agents. A *multi-agent system* is a distributed system where one or more agents collaborate over a network. The collaboration is generally realized by message passing.

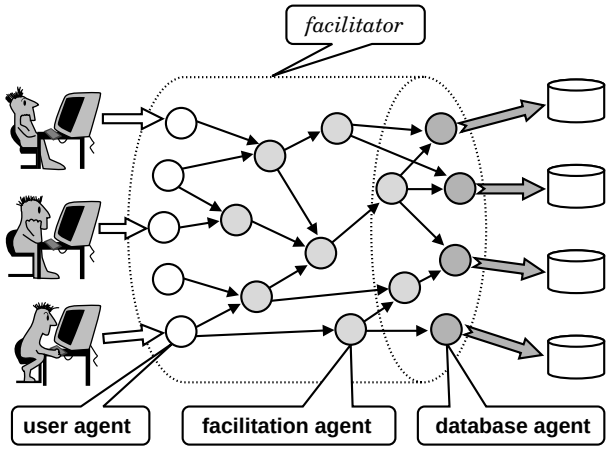

Figure 2
Multi-agent architecture of facilitator.

We show a method to realize the facilitator as a multi-agent system as shown in **Figure 2**. This method solves the issues of the centralized facilitator pointed out in the previous chapter.

Our facilitator consists of the following three types of agents:

User agents: A user agent acts as an interface between a client system of a user and the facilitator. It authenticates the user, transforms the format of a request, and summarizes results of an access.

Facilitation agents: A facilitation agent acts as an intelligent relay of requests. When it receives an access request from another agent, it selects some agents by a simple condition check, and forwards the request to the selected agents.

Database agents: A database agent acts as an interface between the facilitator and a database management system. It controls accesses to the database which it manages, actually accesses the database, and transforms the format of results from the database.

Agents can be distributed over a network. That is, each agent can run on a different host over the network.

Each agent has a set of rules to make decisions on relays of requests. A rule is in the form

{*condition*} → {*agent*}

where {*condition*} is a logical formula in the propositional logic and {*agent*} is the name of an agent. This rule means that

if a received request is consistent with {*condition*} then the request should be forwarded to {*agent*}.

{*condition*} may include access controls as conditions, which controls accesses to some sets of databases by controlling relays. For any database agent, the right side of each rule is the name of the database agent itself.

For example, the rule

[*Bay area*] and [*motors*]

→ `bay.area@tcals.or.jp`

means that

if a received request is about motors in Bay area then the request should be forwarded to the agent `bay.area@tcals.or.jp`.

This is a typical request to find the information on some equipment (motors) stocked in a certain specified area (Bay area).

Each rule of an agent defines a conditional link from the agent to the one specified in the right side of the rule. Therefore, the facilitator can be viewed as a directed graph where each node is an agent and each edge is a conditional directed link from one agent to another.

4. Accessing virtually integrated database

Each access from a user to a virtually integrated database is separated into two phases, "*database navigation*" and "*parallel access*". In the database navigation phase, the facilitator selects databases suitable to a request from a user and, in the parallel access phase, the facilitator accesses the selected databases with the request in parallel. This two phase access enables a user to access the selected databases repeatedly; the user can access the selected databases several times with one database navigation phase followed by several parallel access phases.

4.1 Database navigation

In accessing a virtually integrated database,

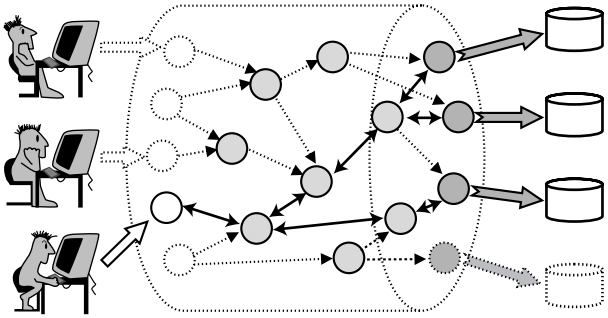

Figure 3
Database navigation.

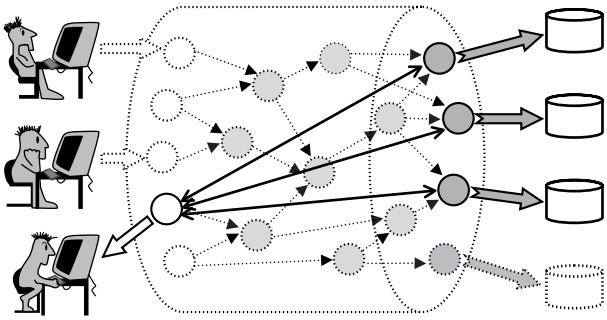

Figure 4
Parallel access to selected databases.

a user is allowed to give a condition on databases to a request. This condition is a requirement of the user on databases to be selected. The facilitator selects databases according to this condition.

In a database navigation phase, agents of the facilitator collaborate by relaying a request from a user with condition checks and select databases suitable to the request as shown in **Figure 3**.

To access the virtually integrated database, a user sends a request to the user agent where the user is registered. This starts the database navigation phase. When a user agent receives an access request from a user, it checks the condition of the request for each rule.

If the condition is consistent with the condition of a rule, that is, the condition of the request and the one of the rule yield no contradiction, then the user agent forwards the request to the agent in the right side of the rule. The user agent carries out this checking and forwarding for all the rules in parallel.

For example, suppose that a user agent has the following set of rules.

[*Bay area*] and [*motors*]
→ `Bay.area@tcals.or.jp`
[*Bay area*] and [*documents*]
→ `Documet.server@tcals.or.jp`
[*Inland Sea area*] and [*motors*]
→ `Inland.sea@tcals.or.jp`

When the user agent receives a request with the condition "not [*Inland Sea area*]", it checks this condition with all the rules in parallel and forwards the request to agents `bay.area@tcals.or.jp` and `documet.server@tcals.or.jp`. Since the condition "not [*Inland Sea area*]" contradicts with the condition of the third rule, the request is not forwarded to the agent `inland.sea@tcals.or.jp`.

When a facilitation agent receives a request from another agent, it also carries out the same checking and forwarding as user agents. A database agent also checks the condition of a received request for each rule. When the condition is consistent with rules, the database agent has its own name as the result. Then, it replies its own name to the sender.

Each agent waits for replies in a pre-defined amount of time. After the reception of the replies from all the accessed agents or after the time limit, the agent unifies and summarizes the replies and returns the summary to the sender. These replies are forwarded along the paths where the request was forwarded. Finally, the user agent receives names of database agents, which correspond to the databases selected by the facilitator as suitable ones to the request.

In this way, the facilitator selects the databases suitable to an access request by condition checking over distributed agents. As a property of this condition checking, the selected databases are consistent with the condition of the request.

4.2 Parallel access

In a parallel access phase, a user agent and database agents collaborate by direct and parallel communications as shown in **Figure 4**.

A user agent accesses the database agents

selected in a database navigation phase. This access is carried out directly and in parallel from the user agent to the database agents; the user agent directly sends the access request to each database agent in parallel. When a database agent receives an access request in this phase, it actually accesses the database that it manages, transforms the format of a result of the access, and directly replies the result to the user agent. After receiving replies from the accessed database agents, the user agent unifies and summarizes the replies and returns the summary to the accessing client of the user. This ends one parallel access phase.

While the purpose of the database navigation phase is to select databases suitable to a request from a user, the purpose of the parallel access phase is to access the selected databases efficiently. Any number of parallel access phases can be repeated for the selected databases in the previous database navigation phase.

4.3 Format of access descriptions

In order to make the system to be widely available from various kinds of clients and database management systems, the system must have a popular communication protocol. The Steel Plant CALS project adopts HTTP as the protocol for the facilitator to communicate with clients and database management systems and, following HTML, prescribes the format of access descriptions, called **FORMAT-X**, suitable to the two phase access to a virtually integrated database. **FORMAT-X** defines the format of messages between client systems and the facilitator and between the facilitator and database management systems.

In **FORMAT-X**, various types of data are defined by using tags similar to HTML in the following way;

<*tag*> {*data*} </*tag*>

where "*tag*" is the name of a type of data. For example, "<USER> Fred </USER>" defines the user name "Fred".

FORMAT-X mainly defines two formats of messages, "*query format*" and "*reply format*". The query format is for messages of access requests from clients to the facilitator and from the facilitator to database management systems while the reply format is for messages of replies from database management systems to the facilitator and from the facilitator to clients.

Figure 5 shows an example of a message in the query format. The tag <SQL-QUERY> indicates that the message is a query to a virtually integrated database. A message in this format has information on the profile of a user (user name, password, affiliation, department, and so on) and on the access control (crypt system, time limit, and so on). The first block of the example indicates this information.

The data indicated by the tag <SQL> is the statement of access to a virtually integrated database. The Steel Plant CALS project also prescribes the database access language, called **VIDAL** (Virtually Integrated Database Access Language), by extending and modifying the standard SQL. This **VIDAL** includes ordinary SQL statements such as SELECT, INSERT, UPDATE, DELETE, GRAND. It also has some **VIDAL** specific statements

```
<SQL-QUERY>
  <USER> {user name} </USER>
  <PASSWORD> {password} </PASSWORD>
  <AFFILIATION> {company code} </AFFILIATION>
  <WORKS> {works code} </WORKS>
  <PLANT> {plant code} </PLANT>
  <MD> {message digest} </MD>
  <TIMEOUT> {time limit} </TIMEOUT>

  <SQL>
    TARGET {condition} FROM {database name}
  </SQL>

  <SQL>
    SELECT {class}.{property} WHERE {condition}
              ...
  </SQL>

  <SQL>
    END-TARGET
  </SQL>
</SQL-QUERY>
```

Figure 5
Query message in **FORMAT-X**.

such as ATTACH (to make a link between instances of two classes) and DETACH (to remove the link made by ATTACH).

Important **VIDAL** statements are TARGET and END-TARGET. These two statements control the database navigation phase. A TARGET statement is in the form

TARGET {*condition*} FROM {*database*}

where {*condition*} is a condition for the facilitator to select databases and {*database*} is the name of a virtually integrated database. When the facilitator receives this statement, it carries out the database navigation with {*condition*} for {*database*}. Then all the **VIDAL** statements after this are in parallel access phases to the same databases selected by the TARGET statement until END-TARGET statement is received.

Figure 6 shows an example of a message in the reply format. The tag <SQL-REPLY> indicates that the message is a reply from a virtually integrated database. A message in this format has data as the result of an access or error messages from agents or database management systems.

5. Maintaining facilitator

The facilitator has no global data and requires no shared memory; only each agent has a set of rules as an agent-specific data. This makes it easier to maintain the facilitator. Modifying the behavior of the facilitator is to modify agents constituting the facilitator and modifying agents is to modify their rules. The condition and the agent name of a rule can be changed. Also, a new rule can be added to an agent. These modifications can be done independently from the other agents.

```
<SQL-REPLY>
  <REPLY-DATA RETURN={the number of instances}>
    <INSTANCE>
    <PROPERTY LABEL={class}.{property} SIZE={size}>
        {data}
    </PROPERTY>
    </INSTANCE>
        …
    <INSTANCE>
        …
    </INSTANCE>
  </REPLY-DATA>
</SQL-REPLY>
```

Figure 6
Reply message in **FORMAT-X**.

Adding or removing some databases for the virtually integrated database causes changing the graph structure of the facilitator. This change may include adding or removing some agents in the facilitator as well as modifying some agents. Again, since the facilitator has no global data, adding and removing agents have no effect to behaviors of remaining agents, that is, they can still work by themselves and, in principle, the facilitator also can work properly.

To add a new agent to the facilitator, the administrator can prepare its set of rules independently from the other agents and then he may ask some other agents to put links to the new agent. When an agent is removed from the facilitator, some links to the agent from others may remain. Although this cause no trouble to those agents who have such links, those links should be removed because they become useless.

For an agent to become alive or dead corresponds to adding or removing the agent in the facilitator. Therefore, changing the status of the agent to be alive or dead does not affect the other agents although the behavior of the facilitator may change. In this way, the maintenance of a virtually integrated database can be localized into neighbor agents. This enables an incremental way of scaling up or down of the system. The facilitator can be scaled up or down incrementally by adding or removing agent by agent.

6. Advantages of multi-agent architecture

Our system has several advantages comparing to systems with centralized facilitators because of the multi-agent architecture of the facilitator.

Load balance: In our system, database accesses are carried out by collaboration of agents distributed over a network. Also, each agent works independently. Therefore, the load of

processing accesses is distributed over agents. Since agents work concurrently, this load balance does not depend so much on the scale of the system.

Agent-local maintenance: Since the facilitator has no global data, each agent can be modified independently from each other. This modification does not require any modification of the other agents. In modifying an agent, the agent may halt but the other agents can work. Therefore, the facilitator works even while some agents are being modified.

Distribution of maintenance and management: Each agent can be invoked in a different machine and can be maintained and managed independently from other agents. This enables each enterprise to maintain and manage its own agents. Therefore, each enterprise bears the cost for maintenance and management of its own agents. Also, each enterprise can put its own policy on its agents, which does not depend on policies of other enterprises.

Agent-based security: Each enterprise can put its own security policy on its managing agents. This policy is reflected over the rules of agents. For example, the policy of access control can be reflected on conditions of rules, which control paths on the directed graph of the facilitator. Since rules are local data of each agent, the information on rules are protected as far as the enterprise manages agents properly. This means that each enterprise has the responsibility for security.

These advantages mean that our multi-agent architecture of the facilitator solves the issues of the straightforward realization of the facilitator as the centralized one. Especially, distributing maintenance, management, and security policy over agents is an important feature in inter-enterprise environments such as CALS.

In addition to the above advantages, our multi-agent architecture of the facilitator produces following advantages as well:

Scalability: The facilitator can be extended by adding agents to and can be reduced by removing agents from the directed graph of the facilitator. As we have mentioned in the above, this does not affect other agents. Therefore, scaling up or down of the facilitator is easy. Also, since agents work concurrently, adding agents does not make the facilitator infeasible. These mean that the facilitator has high scalability.

Robustness: As we have mentioned in the above, even if some agents halt, the other agents can run and therefore the facilitator itself can work. Hence, if we prepare more than one path for each database in the directed graph of the facilitator, then the database is accessible even when some agents halt. This implies the robustness of the facilitator.

In this way, the multi-agent architecture of the facilitator has many properties required as distributed systems.

7. Implementation details

In this chapter, we summarize implementation details of the experimental system of a virtually integrated database developed under the Steel Plant CALS project in Japan.

The facilitator is implemented by **April** programming language system.[10] **April** is a network oriented programming language system, which supports various functions to develop distributed systems over wide area networks such as multi-agent systems. These functions include asynchronous communication, global naming service, multi processes and so on. **April** provides an easy way of message passing between its processes over a wide area network by its asynchronous communication and global naming service. To implement our facilitator, this makes it easy to realize communications between agents. Agents are realized as **April** processes and communications between them are realized as **April** message passing between them. Message passing and names of

agents are managed by **April**.

Figure 7 shows the communication protocols used in the virtually integrated database of the Steel Plant CALS project. In the lower level, **April** communication protocol is used within agents while HTTP communication protocol is used between the facilitator and the outside of the facilitator. In the upper level, **KQML** agent communication language[4] is used within agents while **FORMAT-X** is used between the facilitator and the outside of the facilitator.

KQML is a language for agent communication proposed as a standard for inter-agents communication languages. A message in this format has a performative, which specifies the behavior of agents, and has attribute-value pairs as parameters of the performative. The examples of **KQML** messages corresponding to the examples in Figures 5 and 6 are shown in **Figures 8** and **9**, respectively. In the experimental system, a **KQML** message is realized as an **April** compound data of records and lists.

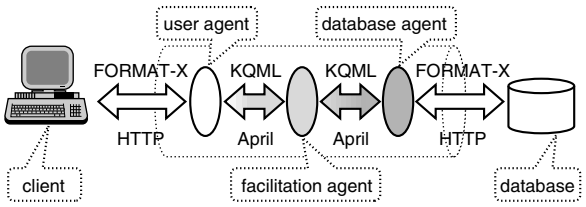

Figure 7
Communication protocols.

```
(_ask_all, [
  (_aspect, DB),
  (_reply_with, {message ID}),
  (_content, [
    (_db_request,
    (SELECT, ([({class}.{property}),... ], {condition}))),
    (_session_id, {session ID}),
    (_user, {user name}),
    (_affiliation, {company code}),
    (_works, {works code}),
    (_plant, {plant code}),
    (_md, {message digest}),
    (_timeout, {time limit}),
    (_request_id, {request ID})])])
```

Figure 8
Query message in **KQML**.

For the facilitator, since agents symbolically process messages to relay and handle results, compound data structure such as lists and records are desired. Therefore, the protocol within the facilitator is realized with the combination of **April** and **KQML**. For clients and database management systems, a popular communication protocol is desired for the system to be widely available. Since HTTP and HTML are widely available over the Internet, HTTP and **FORMAT-X** which follows HTML are in use between the facilitator and the outside of the facilitator. The conversions between these two sets of protocols are handled by user agents and database agents.

The experimental system is distributed over eight sites on the Steel Plant CALS network.[12] These sites are distributed over Japan and are connected by 128 kb/s or 512 kb/s private lines. Each site has a user agent, a facilitation agent, a database agent, and a database. All the databases are SQL databases. In this situation, we measured the performance of the system. **Figure 10** shows the time required to access databases with the numbers of replied data. Although only eight databases were used in the experiments, the result of the experiment shows that the access time would follow a logarithm function of the number of databases as a benefit of the parallel behavior of the facilitator. In each access, most of time was consumed to access databases themselves and, besides the communication overheads, the agent system only consumed less than one second. This partially proves that our system could scale up to a large number of databases.

```
(_reply, [
  (_in_reply_to, {message ID}),
  (_reply_status, OK),
  (_content, [
    (REPLY_DATA, [
       ({class}.{property},...,{data}),... ])])])
```

Figure 9
Reply message in **KQML**.

The Steel Plant CALS project also had a questionnaire to users about the response time of the system. All the users felt that the system worked efficiently enough to satisfy them.

8. Concluding remarks

In the Steel Plant CALS project, only SQL databases are used at all the sites. In real situations, various types of databases are desired to be virtually integrated. Although more researches are required to deal with heterogeneity of database schema,[9] our multi-agent architecture of the facilitator can deal with weak heterogeneity of databases such as format transformation. Each database agent transforms the database own format into the standard one and wraps the transformed data with messages in **KQML**. When a database is replaced by another database that has a different format, it is enough to modify only the database agent.

In order to promote our system more in inter-enterprise environments, adopting standard technologies is important. For wide availability on various platforms, we rebuilt our system in Java,[8] which enables our agents to run in a platform independent way. This version also uses our extended Java runtime libraries, **Kafka**[6] and **Pathwalker**;[14] **Kafka** enables us to program in an agent component fashion and allows components to be extended dynamically by mobile codes. With this function, we realized an on-demand visual monitor of behaviors of the system. **Pathwalker** supports **April**-like network oriented programming facility such as asynchronous communication, global naming service, multi processes and so on. For communication protocols, XML[15] is under consideration instead of **FORMAT-X**. XML is an extension of HTML so that structure of documents can be represented more explicitly by introducing user-defined tags in documents. Since **FORMAT-X** follows the HTML convention, prescribing messages in XML must be not so difficult. As a standard of agent communication languages, the international organization FIPA (Foundation for Intelligent Physical Agents)[5] for the agent technology is proposing the language FIPA ACL. We also adopts this language in our new system and proposes our way of facilitation as a standard of FIPA agent facilitation.

The bilateral CITIS model proposed by the Nippon CALS project prescribes a one-to-one collaboration between a demanding enterprise and a supplying enterprise. Our multi-agent architecture of the facilitator shows that many demanding enterprises can share information with many supplying enterprises. This suggests one possible way to extend the bilateral CITIS model. We are investigating an extension of the bilateral CITIS model according to the multi-agent architecture, which may allow many demanding enterprises and many supplying ones to share information in common.[11]

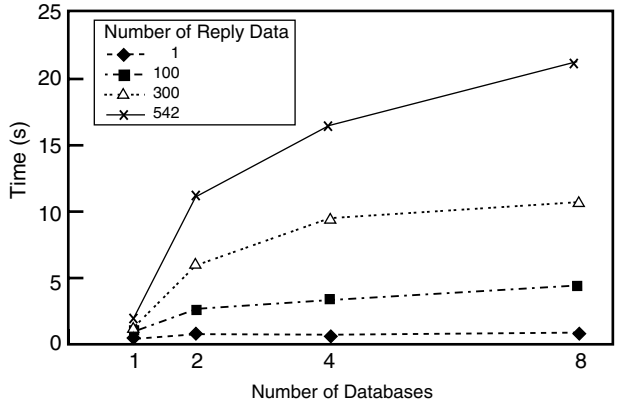

Figure 10
Result of experiment.

Acknowledgements

We would like to thank all the members of the Steel Plant CALS project for their helpful suggestions and comments. We also thank our colleagues in FUJITSU LABORATORIES LTD. and FUJITSU LIMITED. In particular, we gratefully acknowledge Dr. Kazuo Asakawa, Dr. Mitsuhiko Toda and Dr. Francis G. McCabe for their valuable comments and suggestions.

References

1) Y. Asahi: Implementation of information sharing environment with bilateral CITIS. (in Japanese), In CALS Japan '96, pp.9-17 (1996).
2) Y. Asahi, H. Kajihara, T. Ichimura, N. Kiyono, and H. Oono: Information sharing among multiple enterprises through CITIS. (in Japanese), In Proceedings of CALS Expo INTERNATIONAL 1997.
3) CALS Industry Forum, Japan. Proceedings of CALS Expo INTERNATIONAL 1997.
4) T. Finin, Y. Labrou, and J. Mayfield: KQML as an Agent Communication Language. chapter 14, pp.291-316. In Software Agents, The MIT Press 1997.
5) Foundation for Intelligent Physical Agents. See FIPA home page (`http://drogo.cselt.stet.it/fipa`).
6) Fujitsu Laboratories Ltd: Design of multi-agent programming libraries for Java. See Fujitsu's home page (`http://www.fujitsu.co.jp/hypertext/free/kafka/paper/`).
7) M. R. Genesereth and S. P. Ketchpel: Software agents. *Communications of the ACM*, **37**(7), pp.48-53, 147 (1994).
8) J. Gosling, B. Joy, and G. Steele: The Java Language Specification. Sunsoft Series, Addison-Wesley Developers Press, 1996.
9) W. Litwin, L. Mark, and N. Roussopoulos: Interoperability of multiple autonomous databases. *ACM Computing Surveys*, **22**(3), pp.267-293 (1990).
10) F. G. McCabe and K. L. Clark: April - Agent PRocess Interaction Language. In M. J. Wooldridge and N. R. Jennings, editors, Intelligent Agents, volume 890 of Lecture Notes in Artificial Intelligence, pp.324-340, Springer-Verlag, 1995.
11) T. Mohri and Y. Takada: Multi-agent platform for inter-enterprise information sharing. (in Japanese), In CALS/EC Japan 1998, 1998.
12) A. Nishiguchi, K. Shida, and Y. Takada: Steel plant CALS project – business process and information infrastructure across enterprises. In Proceedings of CALS Expo INTERNATIONAL 1997.
13) A. Takekoshi, H. Kato, Y. Horiuchi, and Y. Izumida: Research project to apply CALS to steel production equipment management. (in Japanese), In CALS Japan '96, pp.89-96 (1996).
14) S. Ushijima, T. Mohri, T. Iwao, and Y. Takada: Pathwalker: Message-based process-oriented programming library for Java. In Proceedings of the 11th International Conference on Applications of Prolog (INAP'98), pp.137-143, September 1998.
15) W3C. Extensible markup language (XML). See `http://www.w3.org/XML`.

Yuji Takada received B.A. and M.A. degrees from Department of Behavioral Science and Dr.Eng. degree from Department of Information Engineering, Hokkaido University, in 1983, 1985, and 1993, respectively. Since 1985, he has been with Fujitsu Laboratories Ltd. In 1994 he was a visiting researcher at Department of Computing, Imperial College, U.K. His current research interests include multi-agent systems, distributed computing, groupware, and machine learning. He is a member of EATCS, IPSJ and JSAI.

Takao Mohri received B.E. degree in Mechanical Engineering, and the M.E. and the Dr.Eng. degrees in Information Engineering from the University of Tokyo in 1990, 1992, and 1995, respectively. In 1995, he was a research fellow of Japan Society for the Promotion of Science. Since 1996, he has been with Fujitsu Laboratories Ltd. His current research interests include distributed computing and multi-agent systems. He is a member of IPSJ and JSAI.

Hiroyuki Fujii received B.E. degree in Environmental Engineering from Kyushu Institute of Technology in 1981. He joined Fujitsu Ltd. in 1981, and has been engaged in system engineering, system integration and consultation on controlling process computers, product management systems for the Steel industry. His current interest involves business process reengineering for product management systems. Since 1996, he was a researcher of the Steel Plant CALS project for two years.