November 19, 1999
RT0336
Engineering Technology 10 pages

Research Report

Application Server for the Next Generation of Web Applications

Gaku Yamamoto and Hideki Tai

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Limited Distribution Notice
This report has been submitted for publication outside of IBM and will be probably copyrighted if accepted. It has been issued as a Research Report for early dissemination of its contents. In view of the expected transfer of copyright to an outside publisher, its distribution outside IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or copies of the article legally obtained (for example, by payment of royalities).

Application Server for the Next Generation of Web Applications

Gaku Yamamoto and Hideki Tai

IBM Research, Tokyo Research Lab.
1623-14, Shimotsuruma Yamato-Shi Kanagawa, Japan
{yamamoto, hidekit}@jp.ibm.com

Abstract:

We expect that the next generation of Web applications will provide services based on information about individual users, and will be integrated with e-mail notification services. However, existing approaches are not adequate for developing such applications. We therefore provide a framework and running environment, named Caribbean. The framework is based on a concept of multi-agent systems. Modules, namely agents, are partitioned in accordance with the roles of participants. In this paper, we first explain the type of applications for which Caribbean is intended by describing two examples that we developed. Then, we introduce the Caribbean framework. Finally, we discuss the benefits of the framework for application developers.

Keywords: Electronic Commerce, Electronic Marketplace, Agent, Multi-Agent, Agent Infrastructure.

1. Introduction

Web services have changed from applications providing static information in the early days to applications providing various types of services built on top of the Common Gateway Interface (CGI), servlets, and applications servers. Initially only simple services were provided, but high-quality, complex services such as airline ticket sales and Internet banking services are now available. Some applications provide services customized for individual users, and notification services using e-mail have been increasing. From this background, we can expect that current Web applications will be replaced by complex applications that provide services based on information about individual users, and will be integrated with e-mail notification services. Internet banking services, for example, include not only bank account inquiry services but also notification services that send an alert when the user's bank account falls below some level, notify the user of the latest foreign currency rates, and so on.

In order to develop such applications by using existing technologies such as CGI and servlets, we have to build many functions: not only application logic, but also system functions, such as functions for storing users' preferences and information in a file or a database, performing actions in response to events occurring in a server, and performing tasks even when the related users are not accessing to a server. Moreover, we have no good design principles for developing such applications efficiently and reducing the cost of maintaining programs. As a result, application designers bear a heavy burden.

We propose a design principle for developing such applications, and provide a framework and running environment on top of Java, named "Caribbean." The basic concept of Caribbean is the use of multi-agent technology, in which many agents run on a server and process tasks by using agent communication to cooperate with each other. A typical application of Caribbean is as follows: an agent, which serves a user by holding information and interacting with the user through a Web browser, is created in a server for an individual user. Since the agent can work even when the user is not accessing the agent, it can notify him or her by e-mail of events occurring in the server. Most Caribbean applications access databases and back-end systems. In such applications, the agents that access the databases and the back-end systems reside in a server. The agents that serve their users perform tasks in cooperation with the agents that access databases and back-end systems.

In typical applications, agents that serve users are created individually. Thus, there may be tens of thousands of agents, which could cause a system overload. If a server fails, agents running on the server may disappear. Caribbean's running environment solves these problems by using an agent-swapping mechanism, a thread-controlling mechanism, and an agent recovery mechanism.

In section 2, we present two application scenarios as examples of applications created by using Caribbean. In section 3, we describe the requirements of such applications. In section 4, we introduce related works. Section 5 describes a concept model of applications, and section 6 gives an overview of the Caribbean framework and technologies used in the

Caribbean running environment. The benefits of the framework are discussed in section 6, and our conclusions and plans for future work are outlined in section 7.

2. Applications

2.1. A Providing Service for Travel Information

To give an overview of e-Marketplace systems, we introduce "TabiCan," which is a commercial service on the Internet [2, 3, 4].

TabiCan (http://www.tabican.ne.jp, in Japanese only) is a commercial service site that offers airline tickets and package tours consisting of plane flights and hotel stays. This site hosts travel agencies' virtual shops on the server. Users access this server via their Web browsers and find airline tickets and package tours. They can obtain information from several merchants with a single search action. A user inputs search conditions; for example, "New York" for the destination, "Narita" for the point of departure, May 10 for the departure date, "Japan Airlines" for the airline, and $1,000 for the maximum price. He then pushes the "Start to search" button, and the system queries databases to obtain exactly matched items and recommended items. Recommended items are items that do not exactly match the user's search conditions but that the merchant wants to sell. For example, a merchant agent can offer a ticket on "United Airlines" whose price is only $700. Each merchant has its own selling policy, and is able to customize that policy.

A user's search conditions and search results are kept in the system so that the user can see them again. Even if he switches off his computer, he can see them when he switches the computer back on. This allows users to search repeatedly.

TabiCan provides users with a bulletin board service whereby they can post their requirements on a bulletin board. Merchants can also post their latest product information on the bulletin board. If a product matches a user's requirement, the bulletin board notifies the user of the product information. A merchant can also see the requirements posted on the bulletin board, and provide product information to individual users by e-mail or by showing the information in the user's Web browser when the user accesses the system again.

2.2. A Banking Service

In this chapter, as an application scenario, we present the basic scenario of a prototype that we developed with a bank. This application provides ordinary account, fixed account, and foreign currency account services, which allow a user to:

1. Inquire about an ordinary account, a fixed account, or a foreign currency account
2. Pay money into an account
3. Display foreign currency rates
4. Receive regular notification by e-mail of his or her bank balances
5. Receive notification by e-mail that his or her ordinary balance exceeds the registered limit
6. Receive notification by e-mail that a payment has been made from his or her account
7. Receive notification by e-mail that the specified foreign currency rate exceeds the registered limit
8. Receive notification by e-mail that the profit or loss on a foreign currency account exceeds the registered limit
9. Receive advice about foreign currency accounts

Notifications related to services 5 and 6 are prepared by checking a user's balances regularly. Services 7 and 8 follow the same procedure. Usually the rates of foreign currencies for general consumers are updated at regular intervals, and therefore foreign currencies are checked at the same time. Service 9 is provided by human consultants employed by the bank. A user requests a consultation on his or her financial status. The request is delivered to the consultant, who processes the request at a convenient time and reports the result to the user. A notification of the result is sent to the user by e-mail, since the process may take several days.

3. Related Work

TabiCan provides brokerage, notification and recommendation services. There are many brokerage services; BargainFinder [9] was the first shopping site to provide such a service. Jango [10] can be viewed as an advanced version of BargainFinder. Notification services are provided by Amazon [11] and others. Firefly and NetPerceptions [12] support a product recommendation function. This topic is surveyed in [13]. TabiCan and the banking service described

above also enable that users to keep information in a server. Some sites provide services that enable users to customize the view according to their preferences; however, few services allow users to keep information in a server.

The above mentioned studies focus on services, whereas we focus on an infrastructure for developing servers to provide those services.

4. Requirements

In this section, we summarize the requirements for server programs that support the applications introduced above.

Individual Users' Information Is Kept in a Server

Each user keeps his or her own data, such as preferences and information obtained from a Web site, in a server. A user can see and modify his or her data in the server at any time, using his or her Web browser.

Participants Can Communicate with Each Other

A participant can exchange requirements and information with other participants in a server, even if his or her partners are not accessing the server at that time. The participant can send a message not to just one person but also several people simultaneously. In the example of TabiCan, a user obtains travel information from multiple travel agencies by sending his or her requirements, and a travel agency distributes travel information to multiple users. A reply message may be sent a long time after the request message was sent. In the example of the banking application, a user sends a requirement to a consultant, and the consultant then handles the request at a convenient time and send back a reply to the user.

A Server Catches Events and Executes Tasks for Users

A server catches events occurring at a Web site and executes tasks for users in accordance with the data kept for individual users. Tasks relevant to users who are interested in the events must be executed. In the example of TabiCan, when a travel agency adds new travel information to a database, the TabiCan server is notified of the fact. The server then notifies users who have submitted requests matching the added travel information. In the example of the banking application, when a foreign currency rate is updated, the application server is notified, and then executes tasks corresponding to the wishes of users who are interested in the rate.

Participants Have Their Own Policies for Executing Tasks

A participant executes jobs in accordance with his or her policy. In the case of TabiCan, each merchant has his or her own selling policy to return information on recommended products. A user's task also can be customized according to his or her preferences.

A Server Accesses Back-End Systems

To execute tasks, a server has to connect to back-end systems. The connection may be established either synchronously or asynchronously. In the example of the banking application, users' accounts are managed by back-end systems.

A Server Executes Tasks for Users at Regular Intervals

The server executes tasks for users at regular intervals. Each task is executed on the basis of users' data kept in a server. In the example of TabiCan, each user's information is kept for several days, and is sent to the user by e-mail when the information is removed. In the example of the banking application, the server notifies each user of his or her bank balance by e-mail at regular intervals.

5. Our Approach for Modeling Systems

In this section, we describe our approach for modeling the systems described above. Our approach is based on multi-agent systems that agents execute tasks while communicating with other agents [5, 6].

In our model, an agent keeps its own data and logic. A user has his or her own agent in a server and sends requests to the agent by using a web browser. A service provider does the same. Agents run in a single server and stay in that server for a long time. Most agents wait for message from users. Some agents access back-end systems either asynchronously or

synchronously, receive events from outside programs, or continue to execute tasks on their own threads.

An agent is a self-organized object that does not deliver its object reference to other agents and transfers messages to other agents. The messaging is asynchronous. An agent that receives a message executes a task corresponding to the message. While executing the task, the agent may send a reply message to the sender agent, or may send another message to other agents.

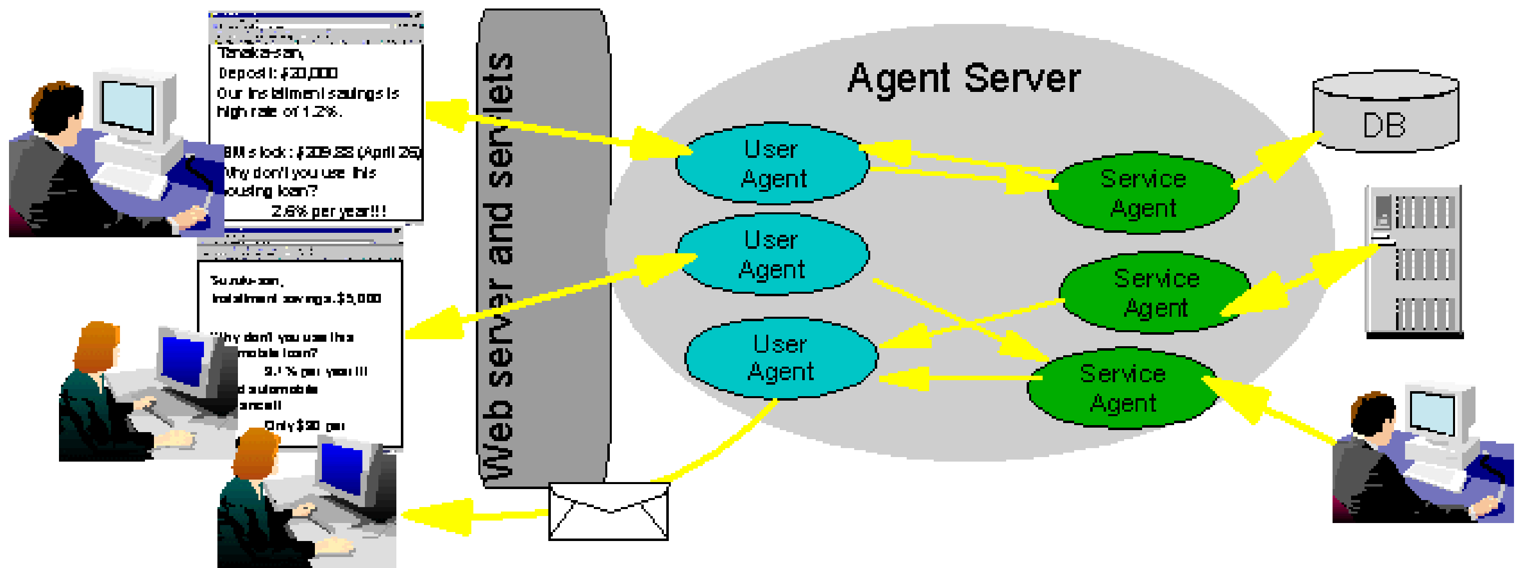

Fig. 1 A System Model

6. Implementation

On the basis of the concept described in the previous section, we developed a framework and running environment, named "Caribbean." Caribbean is built on top of Java. In this section, we introduce the framework of Caribbean and technologies used in its running environment.

6.1. Framework

ObjectBase

A base class of agents is an "ObjectBase" class. An agent is an instance of a class extended from the class. An agent is identified by a unique identifier whose class is "OID." As explained in the previous section, an agent does not deliver to other agent an object reference to itself. Instead, it delivers its identifier. Instead of method calls, an agent sends other agents messages, which are instances of either the "Message" class or a class extended from the "Message" class. An agent program is based on a message-driven programming model. When an agent receives a message, a call-back method of the agent will be called back. The agent executes a task corresponding to the message within a short period.

Context

Agents run in a running environment. The running environment is represented as "Context." There is a single context object in any given running environment. The context provides agent with basic functions, such as creating agents and disposing of agents. An agent has attributes identifying groups to which the agent belongs. Another agent can obtain identifiers of agents belonging to a specified group.

MessageManager

A MessageManager is an object that provides messaging functions to agents. It provides an asynchronous messaging function and a multicast messaging function to agents as fundamental functions. A MessageManager can be extended by an application. For example, an application may need an anonymous messaging function that distributes a message to an appropriate agent in accordance with a message name. A MessageManager object is managed by Context. An agent obtains an object reference to a MessageManager from Context, and sends messages using those messaging functions. Context manages multiple MessageManager objects that are identified by names.

ServiceObject

A "Service Object" is an object that provides agents with services. An agent can obtain a reference to a service object and call it by using a method call. A "Service Object" is an instances of a class extended from "ServiceObjectBase" class which is a subclass of "ObjectBase" class. A service object must register its identifier and service name with a service registry of Context. An agent can then obtain an object reference to a service object.

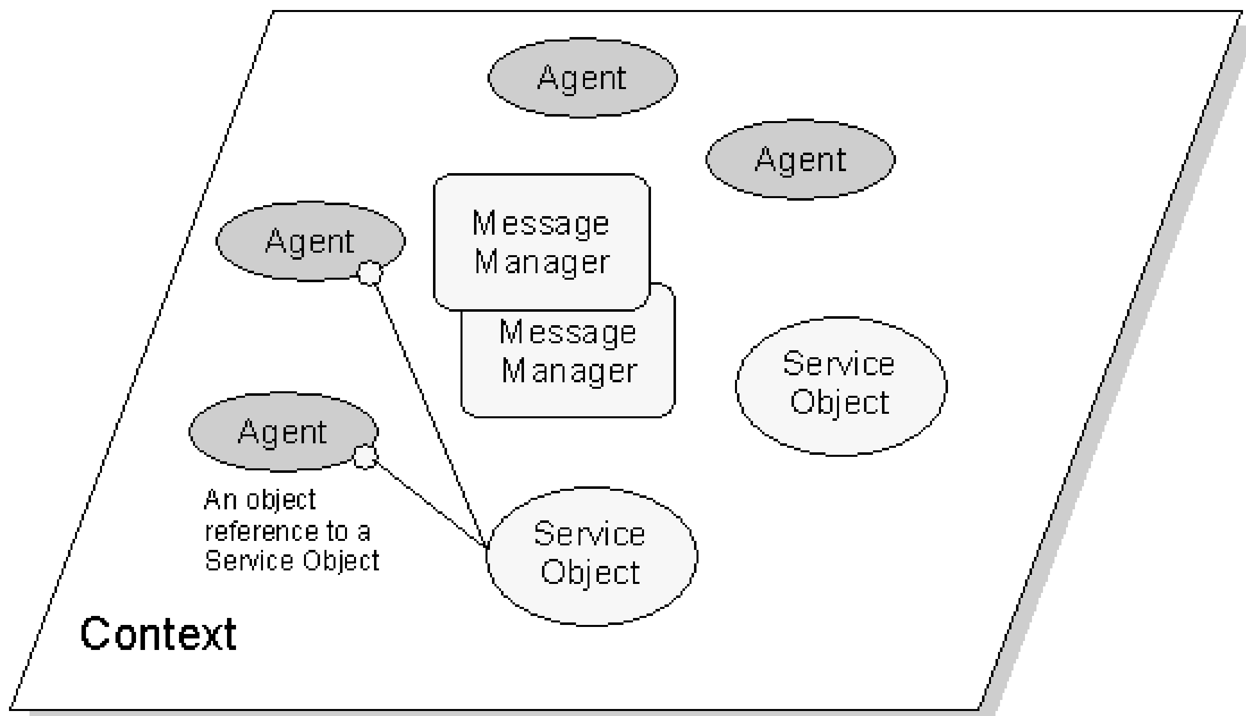

Fig. 2 Overview of Caribbean Framework

6.2. Example of an Application Implementation

We now describe how Caribbean agents are used in an application example. In TabiCan, merchants and users are represented as merchant agents and user agents, respectively. Agents interact to exchange information on airline tickets and package tours. A user instanciates his own agent on the TabiCan server and inputs search conditions. He then pushes the "Start searching" button, and his agent questions all merchant agents that can provide airline tickets. Each merchant agent queries a database several times to obtain exactly matched items and recommended items, and reports them to the user agent. Recommended items are selected according to each merchant's selling policy. After the user agent has received information on airline tickets from all merchant agents, the agent shows the user of the results on his browser.

Merchant agents live during the server runs. User agents live for two days and are removed by the system when their lifetime is over. A user can access his user agent many times while it is alive. Even if he switches off his computer, his agent will still be alive, so he can access it again when he switches the computer back on. A bulletin board is implemented as a "ServiceObject." The bulletin board uses a database to manage user's requirements and product information published by merchants. It also examines which requirements match to newly published product information, and sends messages to user agents corresponding to the matched requirements. A merchant's representative can see users' requirements posted on the bulletin board and send messages containing product information to specified user agents through a merchant agent.

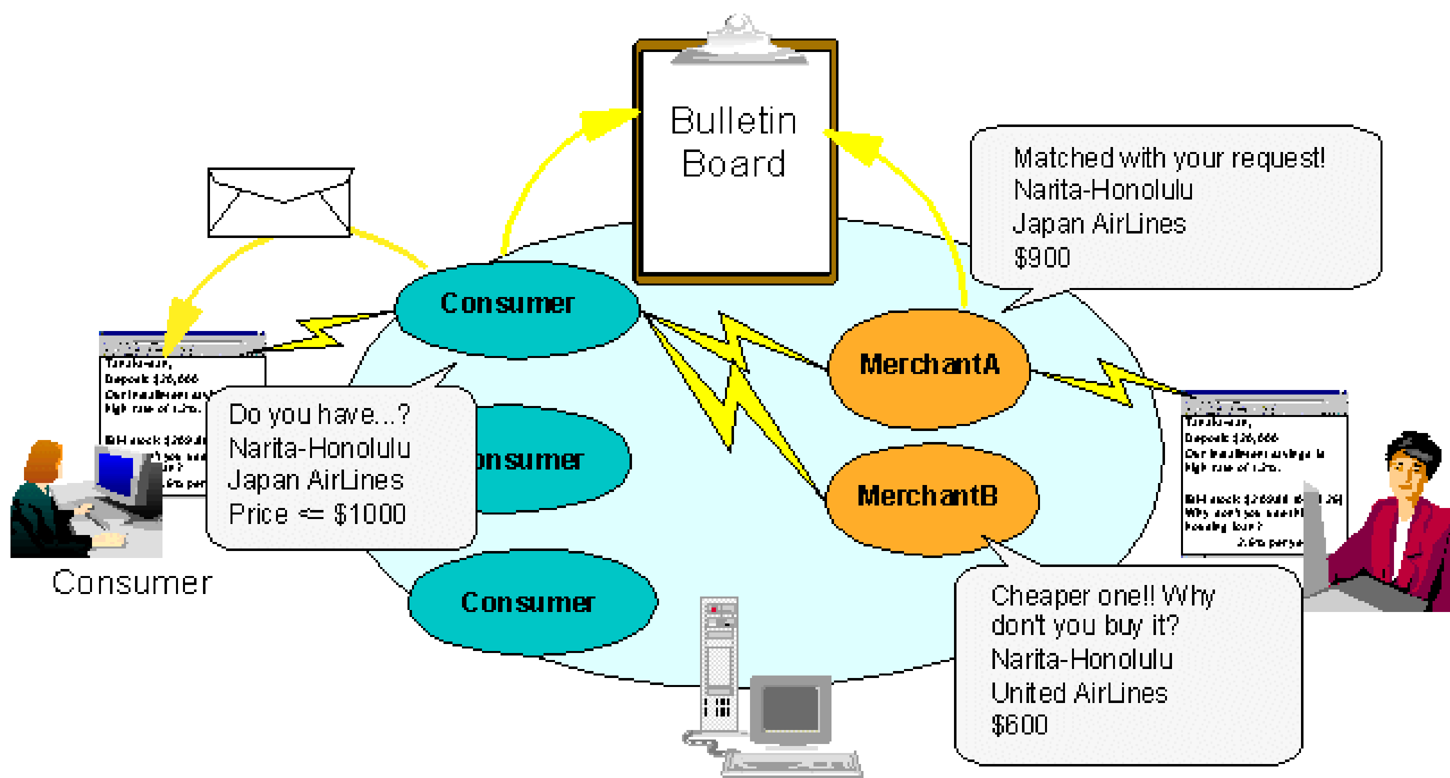

Fig. 3 Implementation of TabiCan

6.3. Technologies

The Caribbean agent server has to manage tens of thousands of agents. In order to manage such a large number of agents in a single server, mechanisms for memory management, thread management, and recovery management are needed; otherwise, the agent server will fail. Since details of these technologies are given in [1], we give only brief summaries of them in this section.

The Caribbean agent server has a mechanism for swapping agents in and out. Some agents can be located in memory and others in secondary storage. If an agent in secondary storage receives messages, then the mechanism reads a memory image of the agent from the storage and activates the agent; this is called "swapping in." At the same time, the mechanism moves another agent into storage; this is called "swapping out." The agent server automatically swaps agents in and out in order to limit the number of agents in memory. The mechanism schedules the swapping to reduce the number of swaps and thus improve the system performance.

The Caribbean agent server has thread pools and limits the number of running threads. It also has a thread scheduler in order to improve the performance. To reduce the number of swaps, the scheduler assigns higher priority to agents in memory than to agents in secondary storage. The mechanism of a thread pool is the same as in transaction processing monitors (TP monitors) [7], however, a thread assignment schedule in Caribbean takes account of agent swapping, unlike a schedule in TP monitors.

The Caribbean agent server has an agent-logging mechanism to take a snapshot of an agent when the latter modifies its data. In order to improve performance, Caribbean supports "group logging" whereby writes snapshots of several agents are taken during a single disk access. It is similar to the "group commit" of TP monitors and database management systems [8].

Agent management policies depend on the types of agents. For example, memory management is required for user agents, since there may be tens of thousands of user agents. On the other hand, service agents can be located in memory, since there are not so many agents. Therefore, Caribbean provides a function for defining a management policy for each type of agent.

Maintenance of programs is a very important issue. If bugs are found in an application program, the program must be changed. To add new functions to current applications, the programs must be updated or changed. Since an agent is a Java object containing a user's data, changing the class of the agent will invalidate both the agent and the user's data. Caribbean provides a function for converting agents of a current program into agents of an updated program.

7. Discussion

There are other approaches to implementing the applications described in section 2. The most typical is to develop modules that implement respective functions are developed. In the case of TabiCan, we developed modules for searching, showing search results, submitting search conditions to the bulletin board, enabling merchants to post product information on the bulletin board, customizing merchant's selling policies, and so on. Several databases are also needed. In the case of TabiCan, there should be databases of users' preferences, product information kept by individual users, product information, and merchants' selling policies.

To compare Caribbean's approach with the existing one, we should consider it from three aspects: system models, method invocation and message passing. In addition, one of the most important features of Caribbean is asynchronous messaging. To compare asynchronous messaging with synchronous messaging, we consider an approach that is similar to Caribbean except that it uses synchronous instead of asynchronous messaging.

System Model

We consider the two approaches in terms of the principles of module partitioning and the relationship between modules and databases. There are two types of information: information shared by all modules, and information owned by a specific module. An example of the former is product information. On the other hand, selling policies and users' preferences are related to individual merchants and individual users, respectively. Therefore, we should consider the approaches from this aspect in addition to the above two aspects.

	Existing approach	**Caribbean approach**
Principles of module partitioning	Based on functions	Based on roles of participants
Relationship between modules and databases	Modules share all databases	An agent covers a database related to the agent
Shared information	Shared by all modules	Shared by all agents
Non-shared information	Shared by all modules	Owned by an agent related to information

Table 1 Comparison of the existing approach and the Caribbean approach

From the above table, we can say that the Caribbean approach is flexible with respect to the addition of new participants, modification of existing participants, and modification of non-shared information. On the other hand, in the existing approach, if a TabiCan system developer changes the scheme of the databases containing users' preferences, the developer has to change many modules, such as those for searching and showing product information. It is also hard to add a new merchant that has its own selling policy, because this might require major modification of an existing search module.

Another feature of Caribbean is that the partitioning of modules is consistent with the roles of participants. This feature makes it easy for system designers and developers to understand a system structure. In our experience, most system designers seem to be able to design a system intuitively.

Method Invocation and Message Passing

The existing approach is based on method invocation. In method invocation, interfaces among modules are static binding. Therefore, when an interface is changed, all modules might need to be compiled. On the other hand, in message passing, interfaces among modules (agents) are dynamic binding. Therefore, even if new messages are added, there is no impact on modules that are not related to the addition. However, problems caused by interface mismatching are detected at run time. To debug such problems, a message trace function is needed. Unfortunately, Caribbean does not currently support such a function.

Synchronous and Asynchronous

Module partitioning in the Caribbean approach is based on the roles of participants. Since participants are on the same level, modules (agents) exchange messages in peer-to-peer manner. This means that not only a user agent but also a merchant agent might send a message at the start of communication. In the situation, synchronous messaging may cause deadlock. The Caribbean running environment has a thread pool and assigns threads to agents in turn. Suppose that the thread pool contains a single thread. In this case, the running environment assigns the thread to the agent, and the agent sends a message to a merchant agent synchronously. However, the merchant agent cannot obtain a thread, since the only

available thread has already been taken by the user agent. The result is a deadlock. In practice, the running environment has several threads; however there is every possibility that deadlock may occurs, because agents exchange massages very frequently and in a very complicatedly. Moreover, the running environment supporting asynchronous messaging can optimize the order of the agent's activities in order to improve performance. Details can be found in [1].

However, if two agents have to execute a task synchronously, an application developer has to manage the states of the agents when using asynchronous messaging. The example applications described above are message-driven applications that do not need synchrony between agents. Therefore, the above drawback is not serious.

8. Conclusion

We expect that the next generation of Web applications will provide services based on information about individual users, and will be integrated with e-mail notification services. However, existing approaches are not adequate for developing such applications. We therefore provide a framework and running environment, named Caribbean. The framework is based on the concept of multi-agent systems. Modules, namely agents, are partitioned in accordance with the roles of participants. Each agent covers its own logic as well as data related to the logic. Therefore, the framework is very flexible with respect to the addition of new participants and modification of existing participants. This will be a beneficial feature in the applications that we expect.

Caribbean can be used in applications accessed by small devices such as small telephones. However, a running environment has to support tens of millions of agents in this situation. A framework and a running environment in a distributed computing environment will be required to support the huge number of users.

9. Acknowledgements

We would like to thank our project members at IBM's Tokyo Research Laboratory and colleagues in other sections of IBM Japan. We also thank Mr. Michael McDonald for checking the wording of this paper.

10. References

1. G. Yamamoto and H. Tai: Architecture of an Agent Server Capable of Hosting Tens of Thousands of Agents, IBM Research, Research Report RT0330 (1999).
2. G. Yamamoto and Y. Nakamura: Architecture and Performance Evaluation of a Massive Multi-agent System, pp. 319 - 325, Autonomous Agents '99.
3. Y. Nakamura and G. Yamamoto: Aglets-Based e-Marketplace: Concept, Architecture, and Applications, IBM Research, Research Report, RT0253 (1998).
4. Y. Nakamura and G. Yamamoto: An XML Schema for Agent Interaction Protocols, IBM Research, Research Report RT0271 (1998).
5. K. P. Sycara, Multiagent Systems, pp. 79 - 92, AI Magazine, Summer 1998.
6. J. M. Bradshow, ed., Software Agents, MIT Press, 1997
7. P. A. Bernstein, and E. Newcomer, Principles of Transaction Processing, Morgan Kaufmann Publishers, Inc. 1997
8. P. A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley Publishing Company, 1987
9. BargainFinder URL: < http://bf.cstar.ac.com/bf >
10. Jango URL: < http://www.jango.com >
11. Amazon URL: < http://www.amazon.com >
12. NetPerceptions URL: < http://www.netperceptions.com >
13. R. H. Guttman, A. G. Moukas, and P. Maes, Agent-Mediated Electronic Commerce: A Survey, URL: < http://ecommerce.media.mit.edu/ >