November 1, 2001
RT0435
Engineering Technology 8 pages

Research Report

An Effective Agent Server for Commercial Portal Sites

Gaku Yamamoto, Hideki Tai

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

ID:128 An Effective Agent Server for Commercial Portal Sites

Gaku Yamamoto
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi
Kanagawa, Japan
+81-46-215-4639
yamamoto@jp.ibm.com

Hideki Tai
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi
Kanagawa, Japan
+81-46-215-5260
hidekit@jp.ibm.com

ABSTRACT

Recent Web Applications provide not only request-response services but also notification services that notify users by email. We have developed an agent server approach with user agents residing on the server side and working for the users. We used the Java-based agent server named Caribbean for two live financial portal sites in 2000. Through this work, we confirmed that the agent server is effective for developing Web applications. In this paper, we describe our experiences with the agent server approach in these applications, showing its effectiveness for application development and its efficient performance.

Keywords

Agent Server, Agent-oriented Programming Model, Web Application

1. INTRODUCTION

Recently, Web applications have become increasingly complex. They provide not only request-response services but also notification services that notify users by email. Moreover, such services have been personalized to perform tasks using individual users' data. One application model to develop request-response services is the DB-centric model that uses a servlet engine as the application server, and a database management system (DBMS) as the users' data repository. However, there are no suitable and natural application models for combined personalized request-response services and personalized notification services. In the DB-centric approach, we may have performance problems because all tasks require DB accesses to retrieve users' data. Even if a system does not have performance problems at first, it might have problems scaling as the number of users increases. If the performance requirements are high, the system requires powerful hardware, resulting in an expensive system.

We have been working on agent server approaches since 1999, and have created a Java-based [1] framework and runtime named "Caribbean". Caribbean is a new type of application server based on an agent-oriented programming model [6]. In the Caribbean model, an agent is created for an individual user and resides at a server. The agent holds the user's data and performs tasks in response to incoming messages. This model can represent in a natural way both request-response services and notification services using an individual user's data. The Caribbean runtime provides an agent persistence mechanism, an agent swapping mechanism which swaps agents between persistent storage and memory, and an agent scheduler to maximize system performance. These mechanisms are needed for a commercial agent server managing many hundreds of thousands of agents. Since Caribbean keeps many agents in memory, it achieves very high performance. We have reported that the performance of Caribbean in our benchmark tests [2] is from ten to over 100 times faster than a Java-based DB-centric system. We used Caribbean for two live financial Web portal sites in 2000.

In this paper, we draw on our experiences to introduce an application example based on these financial portals, and describe the effectiveness of application development and the efficient performance of that application. The example provides personalized request-response services and personalized notification services. The agent server is a new type of application server based on an agent-oriented programming model. Developers can understand the model easily, allowing them to grasp the structure of the complete application. Agent persistence helps them to change the design of the users' data. The overhead of a DBMS can be reduced because an agent server manages the users' data without a DBMS. Furthermore, since an agent server has very high performance, it lowers system costs.

In Section 2 we give an overview of the agent server. In Section 3 we describe an example application scenario based on live financial portal sites that we developed. Our sample implementation of the application is in Section 4. Section 5 is an overview of our design approach. In Section 6, we discuss benefits of the agent server approach and application areas in which the agent server approach shows its effectiveness. Our conclusion and future work appears in Section 7.

2. Overview of the Agent Server

2.1 Agent System

First of all, we give a brief overview of the agent server Caribbean. The agent server is an application server based on an agent-oriented programming model. For a typical Web application, the agent is the object that holds the user's data and performs tasks using that data. The agent reacts quickly in response to messages sent by other agents or by external programs. The agent resides on a server until it is explicitly removed. Each agent has a unique identifier.

An agent exchanges messages with other agents. A message has a message name and parameters consisting of parameter names and values.

An agent uses functions, such as those provided by a database or a mail server, while processing a message. Such functions are resources shared by agents. The resources are implemented as Service Objects. An agent can send messages to a service object or can invoke the methods of a service object. A service object can also send messages to agents.

Agents and service objects can call Context as an interface of the agent server runtime. The Context provides functions for creating an agent, removing an agent, obtaining a list of agents, etc.

The runtime of the agent server manages the agents' activities. In a typical application using the agent server, an agent is created for each user. Therefore, many agents will be created on a server. Though most agents are kept in memory, physical memory size is limited. The runtime monitors free memory space, and when free memory becomes too small the runtime swaps some agents out from memory and writes images of the agents into storage (swap out). When a stored agent is activated, the runtime reads the agent image from storage and restores the agent (swap in). By using this mechanism, an agent server can host an extremely large number of agents beyond the physical memory limitation.

An agent on the agent server is reactive. The runtime assigns a thread to an agent. On the other hand, an agent might be swapped out, required time for disk access. The amount of agent swapping should be minimized. An agent scheduler in the runtime assigns threads to agents in order to maximize system performance.

An agent must survive system failure. When the system restarts, agents must be recovered, so the runtime has an agent persistence mechanism.

Though a single agent server can manage hundreds of thousands of agents, it is best that not all agents be managed in a single agent server. Even if a system has a single computer managing the agents, multiple agent servers should be used on that computer in order to improve performance. Of course multiple agent servers on multiple computers can host very many agents and support very high performance. Therefore, the runtime provides a clustering mechanism supporting multiple agent servers on multiple computers. The client API provided by the Caribbean class library supports clustered agent servers. A client program can send a message to an agent on a group of servers without knowing which agent server hosts the agent.

2.2 Performance

Basically, the agent server keeps many agents in memory. An agent is an object holding a user's data. Since the accesses to a user's data are local memory accesses, they can be performed very quickly. The agent server can achieve very high performance if the server has a sufficiently large memory. Table 1 shows our benchmark results, comparing a system using the agent server with a Java-based DB-centric system. The benchmark is the time for the application to access all of the data for 100,000 users. Each user's data is 50 strings of 10 characters. There are two types of access, read only access, and update access (which includes a read and a write). In the agent server system, an agent is created for each user and the agent holds that user's data. All agents are in memory. In the DB-centric system, the user data is managed by a DBMS. An application server written in Java obtains each user's data from the DBMS using JDBC. The DBMS has sufficient memory to keep all of the users' data in its data cache.

In Table 1, the agent server system is over hundred times faster than the DB-centric system for read only accesses. Even when all accesses are updates, the agent server system is still ten times faster than the DB-centric system.

Table 1. Benchmark Results of the Agent Server System and a DB-centric System

unit: processes per second

	100% read-only	50% read-only 50% update	100% update
Agent Server Approach	22301.50	2077.90	1,069.33
DBMS Approach	168.05	125.40	100.61

3. An Application Example

We used the agent server Caribbean for two financial portal sites providing financial information to users. The application example in this paper is only loosely based on those experiences, because the details of the actual sites are confidential. The example is similar enough to the real sites so that we can discuss the effectiveness of the agent server approach in real systems.

3.1 Application Scenario

The example financial portal site provides financial information, such as foreign currency, loan, and insurance information, through a Web server. The site also provides three notification services through email: foreign currency rates, profit and loss for foreign currency assets, and financial news information notifications. The services are implemented using the agent server.

Notification of Foreign Currency Rates

A user can choose foreign currencies and notification thresholds for those currencies using a Web browser. The foreign currency exchange rates are updated daily. If a foreign currency rate passes a threshold set by the user, the user will be notified by email. For example, a user could set a threshold for the US dollar at 115 yen. If the US dollar rate goes from 114 yen to 115.47 yen, then a notice of the US dollar rate at 115.47 yen will be sent. After notification, the registered threshold will be disabled until the user resets it. The threshold for US dollars is displayed with a mark indicating a notification threshold.

Notification of Profit and Loss for Foreign Currency Assets

A user chooses foreign currencies and registers assets for each foreign currency assets and sets notification thresholds for profit (loss) for each asset. Since the currency rates are updated daily, the profit (loss) of the assets change daily. If the profit (loss) crosses the threshold set by the user, an email notification of the value is sent. For example, a user buys US$1,000,000 yen at 110 yen, and also sets a profit and loss threshold of 50,000 yen. When US dollar rate changes to 120 yen, the profit is +90,909 yen. Since this is over 50,000 yen, the value is sent by email. After the notification, the registered threshold is disabled until the user resets it. The threshold of US dollars is displayed with a mark indicating a notification threshold.

Notification of Financial Information

A user chooses categories of financial news information through his Web browser. When new information for a chosen category appears on the site, email notification will be sent. The email includes only a summary of the information and a URL where the detailed information can be found. The user obtains that information using a Web browser.

3.2 System Requirements

We summarize system requirements for the site in Table 2. The specification is requirements at the initial stage. The system is required high scalability to support a million users.

Table 2. System Requirements

Number of users	Expected to grow to 200,000 users within a few years.
Amount of User Data	At most 8 kilobytes per user
Frequency of Notification	Foreign currency rates and the financial information are updated daily. In the future, the notification processes might run several times per day.
Performance	200,000 accesses per day. At peak times, 1,400 accesses per minute.

3.3 Overview of the System Topology

Figure 1 shows an overview of the agent system topology. In this figure, some components such as firewalls and network equipment are omitted. Although the actual networks are duplicated, it is shown as a single network in the figure.

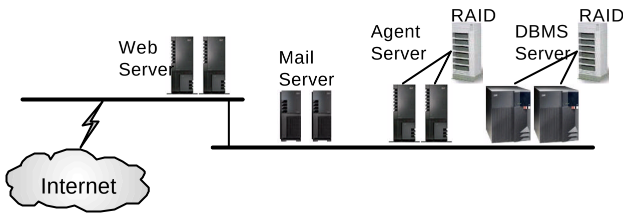

Fig. 1. An overview of the System Topology

This system uses Web servers, servlet engines, DBMS, and the agent server Caribbean. Two Web servers always run on two computers. The servlet engines are on the same computers. Even if one of the computers fails, the system continues to provide services using the other Web server. Two agent servers also run on two computers. If one of the computers fails, the agent server from the computer will start on another computer so that the system continues to provide services. The DBMS is configured on two computers; a primary computer and a backup computer. The DBMS on the primary computer always runs. If the primary computer fails, the DBMS will start on the backup computer.

The system has two dual-ported RAID systems. One RAID is connected to both of the agent server computers and another RAID is connected to the two DBMS computers. The agent server has a persistent agent area on the RAID. Therefore, either agent server can access the persistent agent area if the other agent server computer fails. The DBMS is similarly configured.

3.4 Agent System Implementation

The agent system consists of agents and service objects. Messages are exchanged between the agents and the service objects. The definitions of data exchanged between the agents and service objects are shared among the agents and service objects.

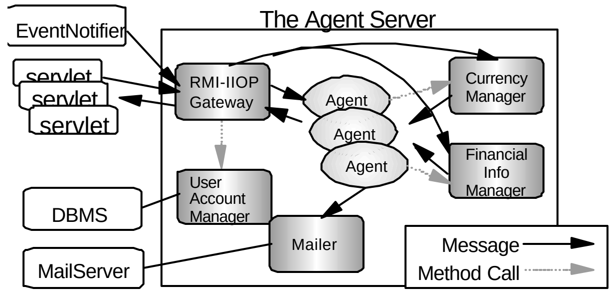

Fig. 2. Agent System Configuration

Agent

An agent is created for each user. The agent has functions for the three services. Since each service is independent from the others, they can be designed independently.

An agent receives messages from a user through servlets and returns response messages to the servlets. Java Server Pages (JSP) processed by the servlets engine generate HTML content.

An agent also handles notification processes when it receives messages calling for DBMS updates. If the agent needs to send email, it calls a service object for sending email.

Service Objects

There are 5 service objects in the agent server:

RMI-IIOPGateway

This service object provides gateway functions between agents and external programs such as the servlets using the RMI-IIOP interface. External programs send messages with agent identifiers to this service object. This service object sends the messages to the agents specified by the agent identifiers. The agents return response messages to external programs through this service object.

UserAccountManager

This service object receives a message with a user identifier and a password from the servlets and authenticates them. If the user identifier and the password are correct, this service object returns a message with the agent identifier of the user. If the user identifier and the password are correct but the user's agent does not yet exist, this service object will create an agent. This service object stores user identifiers, passwords, and agent identifiers in the DBMS.

CurrencyManager

This service object manages foreign currency rate information. Although the information is stored in the DBMS, this service object keeps the information in memory. An agent obtains foreign currency rate information from this service object using method calls. Since the information is in memory, this service object can return the information immediately without accessing the DBMS. When the DBMS is updated, this service object reads the latest information from the DBMS and sends messages to all agents to notify them of the update.

This service object also provides agents with several methods, such as for translating a foreign currency code to a foreign currency name. Since an agent holds foreign currency codes like "NZD" instead of foreign currency names like "New Zealand Dollars" in order to save memory, the agent has to call this service object to translate the code to display its name on a Web browser.

FinancialInfoManager

This service object manages the latest financial information. Detailed financial information is written to HTML files. The agent server sends users summaries of the latest financial information. These summaries are stored in the DBMS. This service object holds in memory the summaries not yet sent to users. An agent obtains summaries from this service object using method calls. Since the summaries are in memory, this service object returns the summaries immediately without accessing the DBMS. When the DBMS is updated, this service object replaces the current summaries with the latest summaries and notifies all agents of the update.

This service object also provides agents with several methods, such as a function for translating an information category code to its information category name. To save memory, agents hold category codes like "AutoIns" instead of category names like "Automobile Insurance", so the agents call this service object to translate codes to display the names on a Web browser.

Mailer

This service object sends email to users. An agent invokes this service object using method calls. The email content is written to disk once. The service object initiates several threads at startup time. Each thread reads the content from disk and calls a mail server (SMTP server) to send the email.

This service object supports multiple email servers for load balancing. If one of the mail servers fails, this service object will use another active mail server. When the failed mail server is restarted, this service object will automatically use that mail server again.

Data Definition

An agent exchanges data in the form of Java objects, with service objects and servlets. Here are some examples:

Currency

Contains foreign currency rate information.
Exchanged between agents and the CurrencyManager.
Member Variables (Name of a value: Type, Description)
curCode: String, a code of a foreign currency
rate: double, the latest foreign currency rate
date: long, date when updated in milliseconds

RegisteredCurrency

Contains foreign currency rate information for display on a Web browser and registration information. This class is extended from the Currency class.
Exchanged between agents and servlets.
Member Variables (Name of a value: Type, Description)
curName: String, name of a foreign currency
threshold: double, threshold for foreign currency notification
registeredDate: long, registration date in milliseconds

FinancialInfo

Contains a summary of financial information.
Exchanged between agents and service objects.
Member Variables (Name of a value: Type, Description)
category: String, a category code
date: long, date when updated in milliseconds
content: String, a summary of notified financial information

Messages

Messages are categorized into three types: messages sent from servlets, reply messages to servlets, and notification messages. Here are some sample message definitions:

CurRateAddReq

Request to register a foreign currency rate notification.
Sent from a servlet to an agent.
Parameters (Name: Type, description)
Currency: String, a foreign currency
Threshold: Double, a threshold for notification

CurRateAddReply

Return code message for a "CurRateAddReq" message.
Sent from an agent to a servlet.
Parameters (Name: Type, description)
Code: Integer, a return code

CurRateGetAllReq

Request to obtain all available foreign currency rates.
Sent from an agent to a servlet.
Parameters (Name: Type, description)
None

CurRateGetAllReply

Reply message for a "CurRateGetAllReq" message. It contains information on all available foreign currency rates.
Sent from an agent to a servlet.
Parameter (Name: Type, description)
List: Currency[], An array of Currency objects

CurRateDBUpdate

Message to notify an agent of a DB update.
Sent from CurrencyManager to an agent.
Parameters
None

3.5 Application Flow

There are two main flows in the application, flows related to a user access, and flows related to notification.

3.5.1 User Accesses

A user invokes a login process by entering a user identifier and password. The request is sent to RMI-IIOPGateway as a message through a servlet and forwarded to the UserAccountManager. The UserAccountManager reads the user's password and finds a user's agent identifier corresponding to the user identifier, and authenticates the user identifier using the password. If the agent identifier is not null, the agent identifier will be returned with a reply message to the servlet. If the reply message does not contain an agent identifier, the servlet will send the UserAccountManager a message to create a new agent whose identifier is stored in the DBMS and returned to the servlet. Since this system has two agent servers, the servlet has to choose an agent server where the agent will be created. The Caribbean client API provides a function to obtain information on a load of each agent server of a clusteres agent system. The load is calculated from information on the numbers of agents in each agent server and the load factor given to each agent server. The servlet creates agents so that the loads of each agent server are balanced.

Once a user completes the login process, the agent identifier is stored with the session information of the servlet engine. Therefore, subsequent requests can be handled without accessing the DBMS until the session information expires. All of a user's requests are transferred to the user's agent as messages through servlets. The agent receives the messages and performs the corresponding processes. During the process, the agent calls the service objects if necessary. For example, if a list of all available foreign currency information is needed, it calls CurrencyManager. The results are sent from the agent to the servlet as a reply message. After the servlet receives the reply message, it invokes a JSP to create the HTML content.

3.5.2 Notifications

The foreign currency database is updated at a scheduled time each day. The update is initiated by a UNIX cron function. After the database is updated, the client program that sends a database update message to CurrencyManager is started. The message is sent to CurrencyManager through RMI-IIOPGateway. When CurrencyManager receives the message, it reads the latest foreign currency rates from the DBMS so the current data becomes the latest data. Then it sends messages to all agents. When an agent receives its message, the agent calls CurrencyManager using a method call to obtain the latest rates for the chosen foreign currencies. The agent verifies whether the rates cross the registered thresholds. It also calculates profits and losses for each foreign currency asset registered by the user. If the agent needs to notify the user, it will call the Mailer with the user's mail address as held by the agent in order to send email.

The notification process for financial information is the same as the notification process for foreign currency rates.

4. Design of the Agent System

The agent server is an application server based on an agent-oriented programming model. Jennings et al. report that an agent-oriented programming model is efficient in developing distributed systems [6]. With support from our experiences, we think that the model is also efficient in modeling internal configurations of the applications providing personalized services through Web interfaces.

Our design approach consists of two parts, agent system design and agent design. Agent system design is the design of the configuration of the agents and service objects in the agent server. In this phase, we also have to define the messages, flows of the messages, service objects' methods called by agents, and data format shared among agents and service objects. Agent design is the implementation design for each agent and service object, and follows agent system design.

4.1 Agent System Design

Agent system design is performed according to the following procedures:

1. Define the participants in the agent system in accordance with their roles.
2. Define agents and service objects corresponding to the defined participants.
3. Define the data shared among the agents and service objects.
4. Define the messages exchanged among the agents and service objects, define flows of the messages, and define the methods of the service objects.

Though a large number of agents may be created in this system, there are not many different types of agents and service objects, nor are the message flows complicated. As Figure 2 shows, the agent system configuration is simple, and so we could do the agent system design without using any formal design methods.

We did have to consider that there could be a large number of agents. This is particularly significant for the notification processes. When the number of agents is in the hundreds of thousands, the number of DB accesses is also in the hundreds of thousands, creating a heavy load on the DBMS and resulting in poor performance. If a message is copied for all agents when DB update is notified, too much memory will be allocated. Therefore, we have to take care about performance and a use of memory when we design the agent system. General techniques for efficient notification processes are described in [4]. The rest of this subsection describes the specific approaches used in our system.

An agent sever computer in this system has sufficient memory to keep all agents in memory. As shown in Table 1, the agent server can achieve very high performance in that case. Moreover, all foreign currency data are updated at once, as is the financial news information. Therefore, we can expect that most agents will be invoked as soon as the information is updated. All foreign currency data and financial news information can be kept in memory. Therefore, we adopted the following approach:

1. When data in the DBMS is updated, the CurrencyManager and FinancialInfoManager read the latest data from the DBMS and hold it in memory.
2. The service objects send messages to all agents notifying them of the DBMS update.
3. If an agent needs to obtain the latest information, the agent calls the methods of the service objects.

This approach has two merits. First, the implementation is simple. Second, the service objects don't need any agent registration information. The notification process is a kind of publish-subscribe model. In a typical implementation of the model, a mediator manages the subscription information, which must be persistent against system failure. This increases system development costs and affects system performance. In the approach that we adopted, the system does not keep any subscription information. Therefore the implementation is simple and it is robust against system failure. Although all agents are invoked, the performance is acceptable because it is expected that most agents will respond to the DB update and performance of the agent server is very good. Although the approach is suitable for this application, we can't say that the approach is the best one for every application.

4.2 Design of Agents and Service Objects

We can use the object oriented design approach to design the internal implementation of the agents and service objects. Since during the agent system design phase, the application is already divided into the agents and various service objects, we don't need to use any large-system design methods for designing the internal implementation of agents and service objects. Moreover, in many cases the specification of a Web application is frequently changed even during development phase. Therefore, we don't use any formal design methods. Instead, we adopted a spiral development approach to implement the agent and each service object. Basically, the internal designs of the agents and the service objects are entrusted to each developer.

A key design point for implementing agents in an agent system that manages a very large number of agents is size of the individual agent. If the agents are large, performance will decrease because fewer agents can be held in memory and the time spent writing agents to disk will be increased. We designed the agents in these systems to have an average size less than 7 or 8 kilobytes. To reduce the size of an agent, agents hold foreign currency codes such as "NZD" instead of foreign currency names such as "New Zealand Dollar." The CurrencyManager provides the function that translates from a foreign currency code into a foreign currency name. Furthermore, we carefully chose the Java classes for the agent's member variables to keep the member objects small. For example, we do not use java.util.Date but simply long for expressing date. We specified the initial size of java.util.Vector and java.util.Hashtable. Keeping such points in mind is important for developing an agent system managing very large numbers of agents.

5. Benefits of Utilizing the Agent Server

There are two main advantages of the agent server: an easily understood programming model and high performance. In this section, we describe these two advantages based on our experiences and the benchmark results shown in Table 1.

5.1 Ease of Development

As described in the previous section, in the first step the designer designs the agent system for the application. At this point the application can naturally be divided into several agents and several service objects. Dividing a whole application into several coarse pieces helps developers grasp the structure of the whole application. Moreover, developers can grasp the structure intuitively by personifying the agents and service objects.

Another important point is the automatic agent persistence. In our experiences, most specification changes are related to end user interfaces. This results in changes in the definition of the user's data. We can deal with such changes by simply modifying the definitions of member variables within the agent objects. For the DB-centric approach, developers have to change the definition of the database with the users' data, which is a time-consuming job. In addition, they have to analyze the effects on all modules. In the agent server approach, modifying the definition of an agent object's member variables is easy and its effects are sealed within the agent itself. Therefore, we can shorten the cycle of the spiral development approach.

5.2 Effects on Performance

One of the important benefits of the agent server approach is high performance. As shown in Table 1, the performance of the agent server is very high. A system to support hundreds of thousands of users requires very high performance, and the agent server can meet this requirement. The agent server also reduces response times for users' accesses. If we developed these applications using the DB-centric approach, the load on the DBMS will be heavy because the application server has to access the DBMS very frequently. In typical Web portal sites providing various services, the data are managed in an integrated DBMS. Though the integrated DBMS reduces system management cost, the DBMS can become a performance bottleneck.

For the example Web site discussed in this paper, the frequency of DBMS accesses by the agent server is not so high because the users' data is managed in an agent server and the data about foreign currency rates and financial news information can be kept in memory by the service objects. Therefore, we can balance the load on the DBMS among all of the services.

Moreover, in the example Web site discussed in this paper, the data about foreign currency rates and financial news information are kept in memory. And the agent server only reads the data. Each agent server in the agent server cluster system can process tasks without interfering with other agent servers. Therefore, the system can increase performance as the number of computers running the agent servers increases. So the system is enough scalable for supporting millions of users.

6. Discussion

When we discuss the characteristics of the agent server, we need to consider it from two aspects; the ease of development and the performance. We could get knowledge that the agent server approach is effective in developing the two financial portal systems. Though the applications we developed are complex from the viewpoint of the application logic, they are not complex from the viewpoint of agent systems. We think this is an important point. The agent-oriented programming model can model a complex application so that developers can grasp the structure of the entire application easily and intuitively. However, we don't think that the agent server is effective for the development of all applications. The point to consider is the types of data handled by the application. The data types can be categorized in two dimensions—the data is shared among all agents (shared/non-shared) and the data is read only or updatable (read-only/updatable).

Non-shared data of both types is data that an agent can efficiently hold in its member variables. An agent server is suitable for dealing with this data from both the viewpoints of ease of development and performance.

The agent server can handle shared read-only data if the volume of data is not too large by using service objects that hold the data in memory. For example, tens of thousands of one kilobytes data objects only require tens of megabytes of memory. Such amounts of data can easily be kept in memory by each agent server in a clustered agent system. In this case, the agent server achieves high performance because all of the data is in memory. However, the agent-oriented programming model does not help to implement any mechanism for managing such data.

If the amount of shared read-only data is large and all of the data cannot be kept in memory, it is not easy to manage such data efficiently. The agent-oriented programming model does not help to implement any mechanisms for managing such data.

An agent server is not efficient for managing shared updatable data. If there is one agent server in a system, the server can manage such data, but when there are multiple agent servers in a system, it is hard to maintain data consistency among the servers.

As already noted, we are not advocating the use of agent servers for all types of applications. However, a system is composed from various parts. An agent server is effective for some of the parts and other middleware such as a DBMS are effective for other parts. By applying an agent server at suitable points, we can reduce development costs and develop a high performance system which balances the system load.

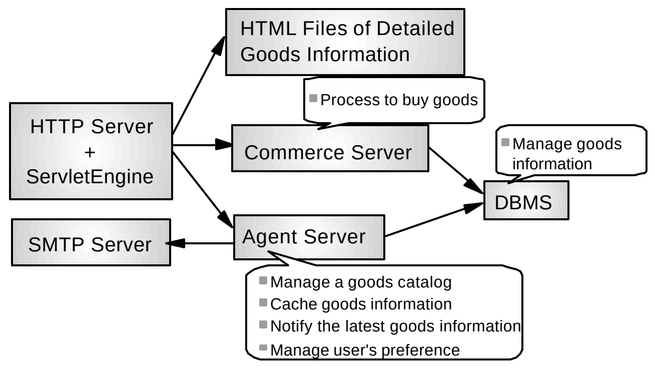

Fig. 3. An Example Commerce Site.

A Commerce Server, an Agent Server, a DBMS, and other servers are integrated into a system. Loads of servers are balanced.

7. Conclusions

In this paper, we described the roles of the agent server, drawing on our experiences in using agent servers for live Web portal sites. The agent server is an application server that achieves high performance by integrating data management with processes and by using memory space efficiently.

The agent server is also an application server based on the agent-oriented programming model. Since the model can represent the application in natural way, developers can grasp the structure of all of the application. Therefore, the development time can be reduced and developers can flexibly deal with specification changes. Our experiences confirm these beliefs.

The performance of the agent server is very high relative to existing DB centric systems. In typical Web portal sites providing various services, the data are managed in an integrated DBMS. Though the integrated DBMS reduces system management cost, the DBMS can become a performance bottleneck. The agent server can reduce loads of DBMS, since the agent server holds users' data.

Though the agent server has the benefits described above, we are not advocating the development of applications based just on the agent server approach. The agent server is most suitable for developing systems that process tasks using each user's data, but it is not so suitable for services that search for items in large volumes of data or for transaction processing services. Such services should be built using a DBMS or a Transaction Processing Monitor. Our position is that by using the agent server with other middleware, we can build better systems (See Fig. 3).

Our experience with agent servers has involved two similar Web portal sites. We hope to verify the effectiveness for other types of applications. Current agent servers have several limitations. For example, if each user data has a lot of data, the performance will suffer. Dynamic load balancing among multiple agent servers is not supported. Because an agent is bound to the agent server where the agent was created, the loads of the agent servers may be unbalanced. We hope these problems will be solved in the future.

8. REFERENCES

[1] G. Yamamoto and H. Tai: Architecture of an Agent Server Capable of Hosting Tens of Thousands of Agents, *Research Report RT0330*, IBM Research, 1999 (a shorter version of this paper was published in *Proceedings of Autonomous Agents 2000*, ACM Press, 2000)

[2] G. Yamamoto and H. Tai: Performance Evaluation of An Agent Server Capable of Hosting Large Numbers of Agents, *Proceedings of Autonomous Agents 2001*, ACM Press, 2001

[3] H. Tai and G. Yamamoto: An Agent Server for the Next Generation of Web Applications, *The 11th International Workshop on Database and Expert Systems Applications (DEXA-2000)*, IEEE Computer Society Press, 2000

[4] G. Yamamoto and H. Tai: Event Distribution Patterns on an Agent Server Capable of Hosting a Large Number of Agents, *Research Report RT0382*, IBM Research, 1999

[5] G. Yamamoto and H. Tai: Agent Server Technology for Next Generation of Web Applications, *4th International Conference on Computational Intelligence and Multimedia Applications*, IEEE Computer Society Press, 2001

[6] N.R. Jennings and M. Wooldridge: Agent-Oriented Software Engineering, in *Handbook of Agent Technology*, J.M. Bradshaw, ed., AAAI/MIT Press