8

Generic Agent Framework for Internet Information Systems

A. Erni, M. C. Norrie, A. Kobler
Institute for Information Systems
Swiss Federal Institute of Technology (ETH)
ETH-Zentrum, CH-8092 Zurich, Switzerland
email: {erni, norrie, kobler}@inf.ethz.ch

Abstract

For effective Internet database services, it is essential that the information requirements of regular users can be met without the typical delays currently experienced using Internet browsers and the World Wide Web. We use co-operating agents to manage both client and server caches, thereby bringing significant performance improvements. The caching and prefetching of information is based on both user and application profiles and agents communicate to ensure the currency of client caches. According to specific application requirements, various forms of agents can be installed on the server and client sides to provide value-added services to both casual and regular users. All component agents are instantiations and/or specialisations of a generic agent. We describe how a specific Internet brokering system for engineering product data has been constructed using our general framework for the development of Internet information systems.

Keywords

Internet Databases, Agents, Web Interfaces.

1 INTRODUCTION

With the development of World-Wide Web (WWW) Interfaces to a number of commercial and research database management systems (DBMS), the task of providing Internet access to databases and their application services is relatively straightforward. For example, the commercial systems Oracle (Oracle Corporation 1996) and O_2 (O2 Technology Inc.), and our own research object-oriented data management system OMS (Norrie 1993, Wuergler 1995, Erni, Norrie June 1997), all provide WWW Interfaces.

However, given the typical delays in WWW access to information currently experienced, it is another step to ensure an effective information service in terms of performance and user satisfaction. In spite of efforts to improve network communications, delays due to network and server overloading are likely to remain with us for some time to come.

To improve performance, we propose an agent-based architecture for the development of integrated, multimedia information services for Internet databases. Our agents execute in the background, cooperating with each other to provide a value-added service in terms of improved user response times for a given Internet information service. The goal of our Internet agents therefore contrasts, and complements, that of existing proposals for Internet agents where the agents provide value-added services in terms of information gathering, filtering and processing. For example, there are a number of proposals for Internet agents to assist in finding information e.g. (Bayer 1995, Voorhees 1994, Shakes, Langheinrich, Etzioni 1997, Doorenbos, Etzioni, Weld 1997), filtering information such as electronic news services e.g (Lang 1995) and also processing information for user specific tasks such as meeting scheduling systems, e.g. (Maes 1994), and electronic mail handling (Payne, Edwards 1995, Maes 1994). Our proposals are similar to many of these in the need for the agents to anticipate user requirements, and also be aware of server characteristics, in order that the desired information can be presented promptly on demand.

A combination of server and client agents provides performance gains to users through active caching mechanisms based on user and application profiles. The agents communicate to ensure the currency of the client caches and prefetch information into the client caches based on predicted user requests.

Casual users may access the information service directly via the usual WWW services. The server agent controls all access to the database and application services and general access is improved by the caching strategy of the server agent. All results of queries frequently requested by clients are stored in a global server cache, thereby reducing response time and database load.

Regular users may further reduce response times, and possibly obtain access to additional services, by registering with the server agent and installing one or more forms of client agent. For registered users, user response times are improved through a combination of techniques to reduce network traffic and perform prefetching of data into the local cache. Cooperation between the server and client agents ensures that only new and updated information is transferred from the server, while other information is accessed via the local cache. Prefetching of data can be based on both user and application profiles which enable user information requirements to be predicted and data to be downloaded into the cache in order that it is available locally when required. We refer to this as *active caching* since it means that data may be cached, not only as a direct result of user-initiated data transfer, but also at the initiation of either a client or server agent.

Since our overall goal is to provide a general framework for the development of Internet information systems, we offer a *plug-and-play* architecture in which various forms of agents can optionally be incorporated to provide improved services. The precise forms of client and server agents used will vary according to both application and user requirements. However, all agents are instantiations and/or specialisations of a generic agent.

The generic agent provides all functionalities required for communication both with other agents, either installed locally or distributed over the Internet, and also with a user. Since a main task of our agents is cache management – whether a global, server cache or local, client cache – an important part of the generic agent is the functionalities required to maintain an active cache.

The various forms of agents that can optionally be installed can be thought of as providing various levels of improved performance to users. The basic system, without agents, exploits the WWW to provide universal access to a given database system and its services. Installation of a server agent, with its caching mechanisms, will provide improved access for all users – casual or regular. By offering a client agent, regular users may register and install a client agent locally with the effect of further improving the performance by means of a local, active cache. If appropriate, a system may additionally offer the option of installing a personal assistant on the client side. This personal assistant maintains the user profile and cooperates with the other agents to increase the prefetching of data according to observed behaviour patterns.

Using this framework, we have developed a number of Internet information systems based on our object-oriented database system, OMS (Norrie 1993, Wuergler 1995) and its WWW interface (Erni, Norrie June 1997). These include an Internet brokering system for engineering product data (IPBS), an integrated tourist information service for snow and avalanche data in the Swiss Alps (Erni, Norrie January 1997) and an information system for renting holiday apartments.

In this paper, we will use the example of the Internet product data brokering system, IPBS, to describe the development and operation of our agents. IPBS was developed in conjunction with the Institute for Design and Constructive Methods at ETH Zurich. The system provides a virtual marketplace for engineers in which they can search for information on products of various companies and follow links to individual company product catalogues.

In the following sections, we discuss our agent-based architecture and generic agent framework in detail. We start in section 2 by presenting the overall architecture in terms of the client and server sides, the agents and the various information sets. In section 3, we detail the functionalities and communication of the generic agent and explain how our plug-and-play approach was achieved. Section 4 presents the Internet product brokering system and a detailed description of how the agents are installed in the context of the WWW is given in section 5. Concluding remarks are given in section 6.

2 ARCHITECTURE

As stated previously, general WWW interfaces have been implemented for a number of DBMS, including Oracle (Ora96 1996), O_2 (O2) and our own research object-oriented data management system OMS (Norrie 1993, Wuergler 1995, Erni, Norrie June 1997). Using these interfaces, Internet access can be provided to information systems either directly (Erni, Norrie June 1997) or through writing HTML pages with embedded queries e.g. (O2). The key issue therefore is not how to provide WWW access to a database system, but how to make it more attractive to both information consumers and producers in terms of performance and ease of use. In particular, we wish to benefit from the open access afforded by WWW, without the drawbacks in terms of poor performance due to network and server overloading and one-hit communication protocols.

It is therefore important that regular users, who may be dependent on Internet information services for their daily work activity, can be provided with improved access over casual users in terms of performance and possibly also functionality. For example, it may be that certain data or services are only available to registered users.

We achieve this through the use of an agent-based architecture as shown in figure 1.

The architecture of a particular agent-based information system can be viewed in terms of its server and client sides with the browser part considered as the user's entry point to the Internet database and its application services. In figure 1, we show all possible forms of agents that we support and the role of each is discussed below. However, as will be discussed in detail later, for any given information system, it is the choice of the system developer, along with the users, as to which agents are actually installed on the server and client sides.

The server side is responsible for controlling all accesses to the database. Because we want to provide access to an existing database system through a single WWW entry point, all available database application services should be integrated into one *Information System* which is accessed via a *Database*

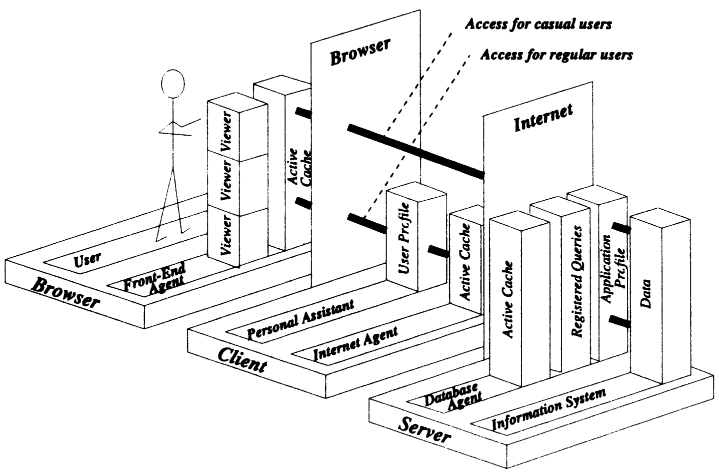

Figure 1 Agent-Based Internet Architecture

Agent. Users access the integrated system through different user groups. We distinguish between casual users, accessing the information services directly via the usual WWW services, and registered users who frequently access the data and therefore have installed local client agents enabling them to optimise their working with the system, see figure 1.

The database agent has its own *Active Cache* storing results of frequently requested queries and therefore reducing query processing. We also refer to this as the system's *Global Cache* since it serves the entire user community. The *Application Profile* stores information about both registered users and data update patterns. This information is used to ensure that only the newest and updated information is stored in the global cache. If a registered user has particular information interests, they may register their interest in terms of the corresponding queries with the database agent. These *Registered Queries* are then used to notify the clients of any relevant changes to the database and help ensure the consistency of the client cache as the database agent can automatically send the newest version of the data to the clients concerned without explicit request.

The client side consists of two cooperating agents referred to as the *Personal Assistant* and the *Internet Agent.* The personal assistant keeps track of a *User Profile* where knowledge about user behaviour and actions to be performed is recorded. The profile may be either specified directly by the user or generated by a learning component of the personal assistant based on the monitoring of user actions. The personal assistant communicates with the Internet agent to initiate prefetching of query results in the user's local cache. Such a trigger can be either a time trigger, for example if a user requests some information every day at 5pm, a data update operation on the server side or a specific action performed by the user. The personal assistant also communicates with the user to inform him of actions taken on his behalf and whether information of interest has changed as reflected by updates to the local cache.

The main task of the Internet agent is to reduce access times for the user. The user response times are improved mainly by active caching, which is achieved in cooperation with the personal assistant and the database agent. Active caching is based on both user and application profiles. Based on the user profile, the personal assistant predicts when the user needs which information, and requests that the Internet agent loads this information into the local cache. Based on the application profile, the database agent knows which data has been updated and sends the new version to the Internet agent, which then stores the information in the user's local cache, also referred to as the *Active Cache.*

The user interacts with the information system through the browser component which is implemented in Java and is made available to the user through a standard WWW browser. The application front-end comprises a browsing component and an agent referred to as the *Front-End Agent.* This agent employs different *Viewers* for the different presentations of the data and has its own *Active Cache* to improve performance while browsing back and forth in an information system. With all forms of agents installed, the information system may therefore have four types of agents and three levels of active caching as shown in figure 1.

Having described the general architecture and the various forms of agents, we now discuss the different architectural variants possible. These variants reflect the various levels of cooperative caching and personalisation that may be supported depending on specific application system requirements. Important is the fact that each level of support corresponds to the installation of a different form of agent – and further agents can be installed dynamically and transparently with the only visible effect being improvements in performance, and possibly also functionality.

A driving force in the development of our framework is the current trade-off that arises between universality and efficiency of Internet access. Through WWW, casual users are able to access an information system easily, and without having to install any special software locally. However, WWW access is typically much slower than traditional client-server access over stable network connections with the effect that regular users are soon frustrated. Regular users would be only too willing to download and install software in return for improved performance – especially if the means of access remains unchanged.

In the following figures we show how our generic agent framework can be used to put an Internet information system together and achieve an architecture according to the characteristics and requirements of the different user groups. In figure 2, the basic architecture to support an Internet information service is shown.

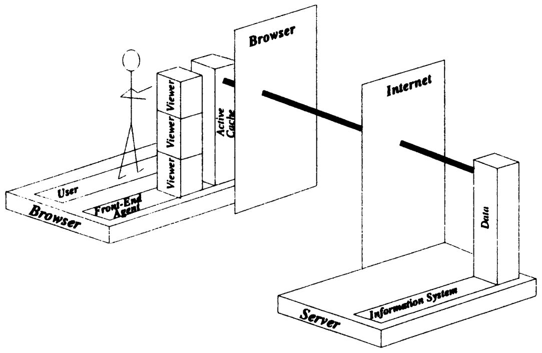

Figure 2 Basic Agent-Based Internet Architecture

Here, the only change from direct WWW access to the database concerns the use of an application-specific interface as supported by a front-end agent. The user accesses the data from the information system directly without the help of any other agents. Every query is evaluated by the database on demand. Therefore even if many clients need the same query, it has to be evaluated each time.

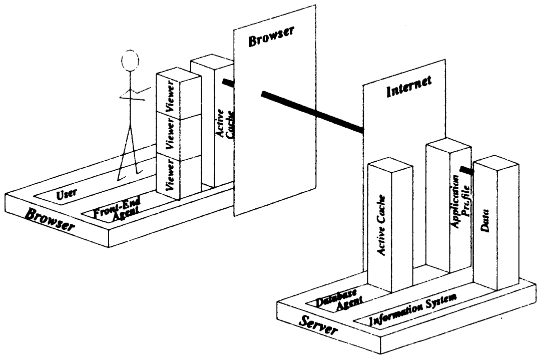

Figure 3 Improved Agent-Based Internet Architecture

The next level of enhancement which can be used is to install a database agent on the server side as shown in figure 3. This architecture improves performance for all users through the caching of query results. For any query, the database agent first checks if the result is already stored in the global cache. If so, the query does not need to be evaluated again, and the result is directly sent back to the user. Note that this contrasts with the caching policies of many proxy servers which tend not to cache dynamically generated pages. In fact, it is frequently the case in information systems that top-level queries used to navigate the database, or inspect new information, are requested by many users – and repeated by individual users. This strategy of query caching alone leads to significant improvements in access to Internet databases – and is, in fact, now used in all general access to our own Internet database system, Internet OMS (Erni, Norrie June 1997).

In all variants of our agent-based architecture with a database agent on the server side, from the point of view of casual users, the architecture will be as shown in figure 3. However, regular users may optionally register and then download and install one or more local agents resulting in the architectural view shown in figure 4.

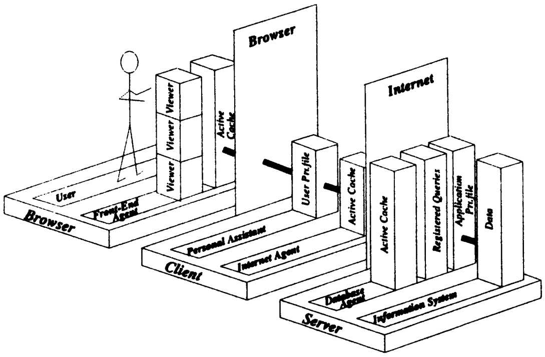

Figure 4 Agent-Based Internet Architecture for Regular Users

Faster access to the desired data can be achieved in two steps. First, by installing only the Internet agent which manages a local active cache, thereby reducing network traffic. Second, by installing a personal assistant which learns from user behaviour and, in cooperation with the Internet agent, initiates transfer of data into the local cache in anticipation of user actions. As explained previously, cache coherency is ensured through cooperation between the Internet agent and the database agent.

We emphasise this step-wise enhancement of a system, to show that agents are developed as component software which may optionally be introduced into the system for improved performance – without affecting how the user accesses the system. Within any one system, different users may have different architectural views depending on whether or not they are prepared to register and install software locally. The agents themselves are configured based on the generic agent described in the next section.

3 GENERIC AGENT

The goal of building a generic agent is to allow a developer to easily put together an Internet information system through the specialisation of component objects according to the requirements and characteristics of the application system.

Each component agent is an instantiation, and possibly specialisation, of a generic agent. The main task common to all agents is the ability to communicate both with other agents and with users. Further, most agents are concerned with the management of a local active cache. The exception to this is the form of agent known as a personal assistant.

Therefore, the two main functionalities of our generic agent are communication and cache management. The generic agent is implemented as a Java class and specialisations of it can be achieved by adding or overriding some of its methods.

A system may comprise many forms and instances of agents resulting in a complex network of cooperating agents. For example, in figure 5, we show a three-level architecture built from database, Internet, personal assistant and front-end agents.

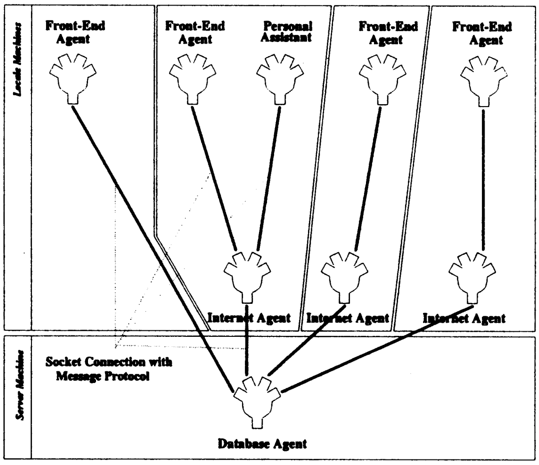

Figure 5 Communication between specialised agents

Communication between agents may be local or across a network. In all cases, communication is via sockets using a self-defined message protocol. At the bottom of figure 5 is the database agent installed on the server machine. It accepts requests from many Internet agents and also directly from a front-end agent, each of them running on a client machine and communicating with the database agent via raw sockets. An Internet agent itself accepts requests from front-end agents, database agents and, in one instance shown in figure 5, a personal assistant.

Each agent may have several client agents – but at most one server agent. This leads to a tree-structure as seen in the example of figure 5. Thus, each front-end and personal assistant has only one Internet agent as a server, and each Internet agent has only one database agent as a server. However, a database agent can serve many Internet agents and an Internet agent could serve many front-end agents and personal assistants.

The generic agent is able to create socket connections, either locally or via the Internet, and is always ready to accept new ones. The generic agent maintains these connections and informs the other agents if a connection breaks or if an agent is about to shut down.

The general data flow goes from top to bottom, that means from the clients (front-end agent or personal assistant) to the server (database agent). Agents communicate by means of message packages the structure of which is shown in figure 6.

Package ID	Package Type	Caching Strategy	Time-Stamp	Data (Query, Answer, Special Command)

Figure 6 Structure of a Message Package

A message can either be a database query or a special agent command. The *Package ID* is used to identify the outgoing query with the incoming answer. The *Package Type* defines the type of the data, if it is either a database query, a special agent command or an answer. The *Caching Strategy* indicates whether the data received is to be stored in the active cache of the agent. For example, special agent commands or error messages from the database should not be stored, however valid query results are stored. The *Time-Stamp* is necessary to keep the active cache up to date. In storing the time the query was evaluated, a detection of query results which are out of date is possible.

To be able to handle this kind of message, the generic agent has built-in functionalities to accept queries from client agents, to process queries by either taking the result from the active cache or forwarding the query to its server agent, to wait for results from the server agent and to send the results back to the client.

The main task of the generic agent is to reduce query response time for the user. To achieve that the generic agent has to organise and maintain an active cache. The generic agent has functions to store and delete data in its cache and also to automatically refresh data by sending a request to its server agent when it receives notification that the data has been updated in the database.

Another important feature of an agent is communication with the user. The generic agent provides a graphical user interface through which the user can configure the agent by specifying various parameters such as the host and port number of its server agent. Through this interface, the user can also view the log file detailing the agent's actions and various statistics about the agent's activities, such as the average processing time for queries and the hit rate of the cache.

Having explained the different functionalities of our generic agent, we know that the agent receives messages from its various clients and from its server. The agent has to process these messages, maintain its cache, listen for new clients requesting connections and display its actions to the user. To optimise performance, the agent decomposes all of these actions into different tasks which are executed in parallel. Figure 7 shows the architecture of the generic agent, consisting of the various tasks.

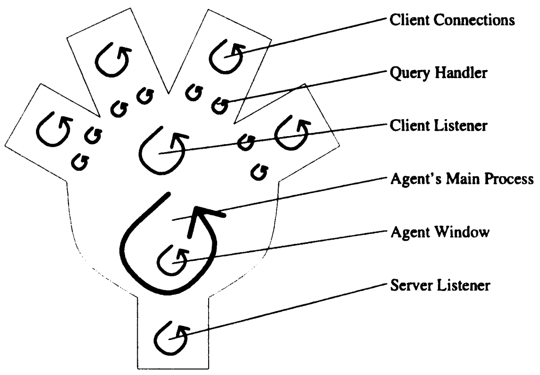

Figure 7 Architecture of the Generic Agent

In starting the agent, a background process is created, shown as the *Agent's Main Process*, which is responsible for the administration of all other subtasks. At the same time, the user interface, called *Agent Window*, is presented to the user which enables him to initialise the agent and to observe all actions the agent is executing. See also figure 8 a).

With the initialisation of the agent, the *Client Listener* process is started, which listens for new clients trying to connect to the agent. At the same time, the *Server Listener* process is initiated, which connects to the user-specified server agent and listens for messages arriving from there, see figure 8 b).

If a connection request arrives from a client, the client listener starts a *Client Connection* process, which handles all messages to and from that client, as shown in figure 8 c). By creating a separate process, the client listener is ready to accept other client connections without affecting the client connection just established.

If one of the connected clients sends a query, a *Query Handler* process is created which processes the query by checking for the result in the active cache or sending the query to the *Server Listener* process which forwards it to the server agent, see figure 8 d). While the query handler is processing the query, the client connector is ready to accept other queries.

Decomposing these tasks into different parallel processes enables many queries from different clients to be processed quickly, thereby providing all users with a fast and effective Internet information service.

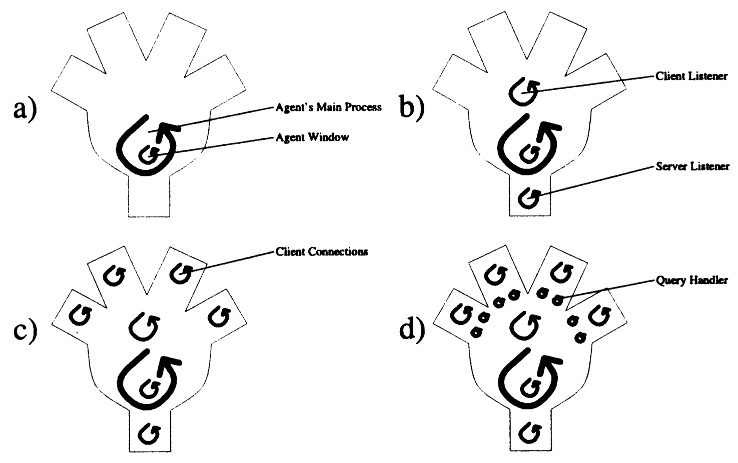

Figure 8 Tasks of the Generic Agent

Through the implementation of this generic agent, the main functionalities of all forms of agents used in our agent-based Internet architecture are provided. From the generic agent, we have developed generic forms of the database, Internet, front-end and personal assistant agents. In the next section, we describe how each of these was developed from the generic agent by considering the requirements and operation of a specific application system IPBS.

4 INTERNET PRODUCT BROKERING SYSTEM

The goal of the Internet product brokering system, IPBS, is to provide engineers with Internet access to a database of information about various companies and their product catalogues. As part of the design process, engineers will perform an initial search for products meeting their requirements and, on identifying likely sources, follow Internet links to the catalogues available on individual company Web sites.

An earlier prototype system was developed by our project partners, the Institute of Construction and Design Methods at ETH Zurich, based on a relational database management system and using HTML pages with CGI scripts. While the system provides the basic functionality required, it is slow and navigation through the database tends to be tedious, since a user must always start from the same top-level entry point and navigate to their chosen data. We implemented a new version based on our own object-oriented database management system OMS, its WWW interface and our agent framework. The resulting system has proved more flexible in terms of navigation and also much faster.

We use this application system to describe how the various forms of agents in our framework are related to the generic agent described in the previous section. Using the framework, an application system can be rapidly constructed using the *plug-and-play* approach – with only some cases requiring specialisation of the agents for extended functionality. In this section, we detail how the various forms of agents were developed from the generic agent by describing how the IPBS system could be developed starting with only the generic agent.

The first stage is to develop the underlying database system containing product and company data using the OMS object-oriented database management system (DBMS). OMS provides many features, such as data model and query language expressiveness, browsing capabilities, uniform handling of data and metadata and also URLs as base types, that are beneficial to this and other applications. However, the agent framework is not dependent on the use of OMS. Any database management system could be used, provided that there is a means for communication with an agent so that the agent can forward relevant queries and also be notified of changes in the database – either directly or through a monitoring capability. In practice, the only parts of the system which are DBMS dependent are the application front-end interface where queries are formulated and the means of communication between the database agent and the DBMS.

The implementation of the database agent is straightforward as most of the functionalities are already provided by the generic agent, however a few extra functionalities specific to the tasks of the database agent have also to be provided. In this way, you can consider that, as a specialisation of the generic agent, we implement a generic database agent. This agent is, in turn, used to implement a specific database agent for IPBS capable of interfacing with the OMS database system.

As described in the previous section, the generic agent provides a message protocol which enables communication with other agents. This message protocol has to be extended such that communication with the database is also possible. Additionally, a feature has to be implemented which allows the database agent to observe all changes of data made in the database. This is especially important for keeping the cache up to date and informing all Internet agents about changes to data. In connection with cache maintenance, the database also has to handle registration of both users and queries. This feature also has to be implemented.

The main functionalities of the Internet agent are maintenance of the cache and routing of messages to the correct destination. These functionalities are all provided by the generic agent, and the Internet agent can be implemented directly using the generic agent.

As described in section 2, the personal assistant keeps track of the user profile, where knowledge about user behaviour and actions to be performed is recorded. The profile may be either specified directly by the user or generated by a learning component of the personal assistant based on past user behaviour. In IPBS, the learning component is not required as the user specifies the data of interest directly. The main task of the personal assistant is therefore to ensure that the defined data, as requested by the user, is always stored and current in the user's local cache. Another task is to inform the user of any changes to the data of interest. To be able to attract the attention of the user for such announcements, we visualise the agent as shown in the upper left corner of figure 9. Different facial expressions and a dialogue box are used to inform the user of changes to data and actions of the agent, for example, whether the agent is idle, working or halted.

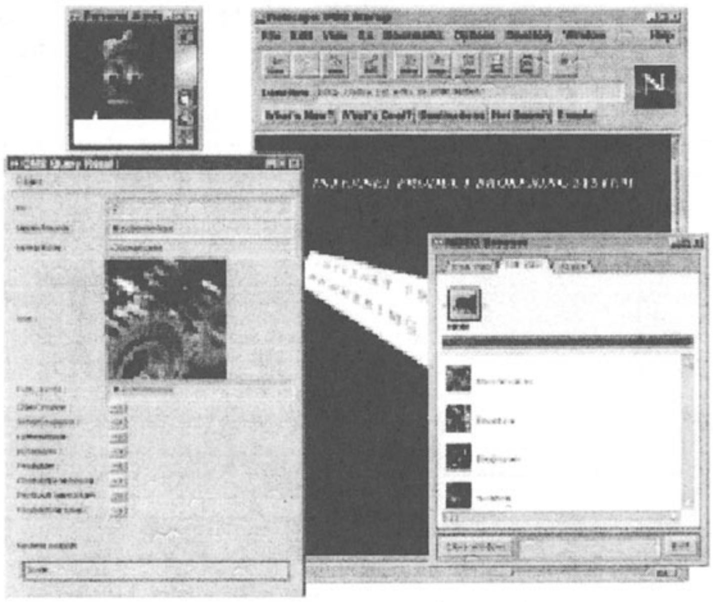

Figure 9 Internet Product Brokering System

The front-end agent controls the user interface and runs as a Java applet inside the browser. Because it is responsible for the visualisation of results and for user's requests, the generic agent has to be extended to support these features. This is done by implementing different viewers for the different types of data to be presented to the user. Two of these viewers for IPBS can be seen in figure 9. The viewer on the right was built specifically for IPBS and is used to browse through the product hierarchy searching for specific product groups. On the left side, a dynamically generated viewer is shown. This viewer is used to display any OMS objects on the client side. Depending on the parameters sent by the database, this viewer automatically generates the appropriate window format for the corresponding query result. This viewer is not application specific and, in fact, is the basis for the WWW interface to OMS provided by

Internet OMS (Erni, Norrie June 1997). By clicking on the various buttons of the displayed objects, the next query is sent to the database and the result is again presented in a dynamically generated window format. In this way, an application specific interface may be developed rapidly using a combination of the generic viewers for displaying database objects and application specific viewers.

As described in this section, it is relatively straightforward to implement the various agents of the framework based on the generic agent. In practice, the IPBS system was developed from this framework. Details of IPBS and its implementation are given in (Ruser 1997). The main task was to design the application interface and decide on the role of the personal assistant in terms of how much personalisation was required and how this should be provided. In IPBS, the personal assistant task is minimal since no monitoring or learning component is involved. In such cases, we actually recommend integration of the personal assistant with the Internet agent to reduce the number of communication layers between the user and the database, thereby enhancing the overall performance. However, we also have experience in developing other applications with more sophisticated personal assistants which construct user profiles based on the monitoring of user behaviour e.g. (Honegger 1997).

5 AGENT INSTALLATION

In this section, we describe the installation of the system in terms of how and where agents are installed and how the start-up and registration process functions. Specifically, we present our solution to how agents can communicate with the application Java applets given the various Java security restrictions. Further, we describe how this solution effectively allows us to combine universality of WWW access with the performance benefits of client-server access over stable connections.

Figure 10 shows the installation of agents from the perspective of a casual user. The database system and the server agent have to be installed on the server machine, as shown in the right of figure 10. For a user to access the application system, two HTML documents are needed and these are also stored on the server machine. One HTML document is the startup page containing a startup applet needed for the initialisation of the system, and the other HTML document contains the front-end applet and the viewers, which then builds the user interface to the system.

The client machine should be viewed as two logical machines — the local machine on which local applications execute and the Java virtual machine within the browser on which Java applets execute. This distinction is made since security restrictions prevent Java applets that have been downloaded and execute within a browser to communicate with local applications. However, we note that with recent developments in secure and signed applets as available in the first release of Java Security in JDK 1.1 (Sun Microsystems), it should be

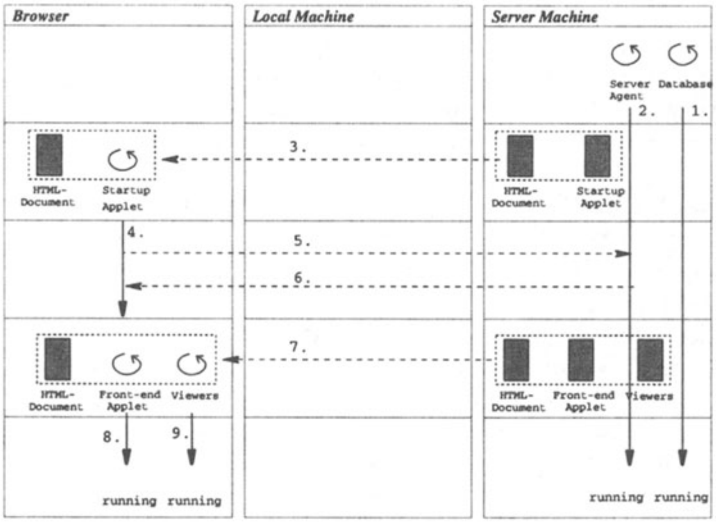

Figure 10 Startup-Process for Casual Users

possible to avoid these restrictions in the future. In the case of a casual user accessing the database, no software is installed on the user's local machine and hence there are no components of the application executing on the local machine as shown in figure 10.

In figure 10, we also show the steps involved in installing the application and accessing the database. Installation of the system involves starting the database server (1) and the server agent (2). A user accesses the system by downloading the HTML-document containing the startup applet (3) which then executes and starts the application locally (4). Because the server agent keeps track of all Internet agents running, the startup applet first establishes contact with the server agent (5) and (6). Since this contact is important in the case of registered users, we describe it in more detail later. The startup applet then downloads the HTML document (7) containing the front-end agent (8) and the viewers (9) from the server machine and the application is started.

In contrast, figure 11 shows the installation of the agents from the perspective of regular users who register and install client agents on their local machine.

In this case, a copy of the HTML document containing the front-end agent and the viewers is also stored on the user's local machine. This is necessary because of Java's applet security restrictions. Java applets are only allowed to connect with the machine from which they were downloaded. This means that they cannot connect to local applications. Further, they cannot write to local files. There would therefore be no way for the front-end agent and Internet agent to communicate directly. However, by storing a local copy of the applet for the front-end agent on the client machine, we can switch the downloaded

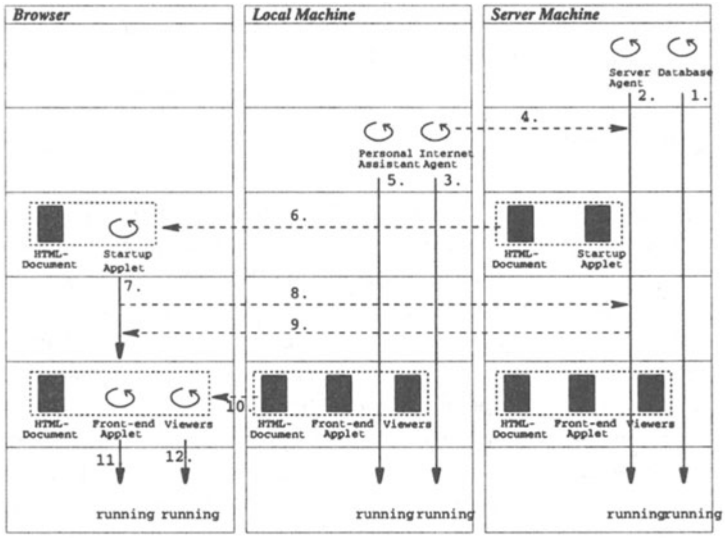

Figure 11 Registration- and Startup-Process for Regular Users

applet with the local applet, thereby allowing the applet to communicate directly with the local Internet agent.

As before, the database system (1) and server agents are started on the server machine. On the user's local machine, the Internet agent is started (3) and it first sends the local address of the HTML page containing the front-end applet to the server agent (4), which registers the Internet agent and stores the address received. Next the personal assistant is started (5). The user accesses the application system in exactly the same way as the casual users, i.e. by downloading the HTML document (6) containing the startup applet which then executes (7). The startup applet contacts the server agent (8). In this case, the server agent knows of the existence of a local Internet agent and it sends back a message giving the address of the local HTML page containing the front-end agent (9). This HTML document is then loaded from the user's local machine enabling the front-end agent to communicate with the Internet agent. The applets for the front-end agent and viewers are then started and the application interface presented to the user is exactly the same as with the casual user.

The solution presented enables us to provide uniform WWW access to the application system for all users. However, in the case of registered users with locally installed software, the server agent uses its knowledge of the user to effectively switch from running the application via WWW to running the application via a traditional style client-server architecture over socket connections. In this way, not only does the performance of the system improve due to the use of active caching, but also due to the system running over stable socket connections rather than HTTP.

6 CONCLUSION

We have presented a framework for the development of Internet information systems based on an architecture that uses cooperation between client and server agents to improve user response times through active caching mechanisms. This framework is based on a generic agent containing all functionalities needed to cooperate with other agents and with the user. Based on this generic agent, we provide various forms of client and server agents from which specific Internet information systems can be built.

Our experience in building various application systems using this framework proves that such systems can be developed quickly and that they provide far better performance than simply using normal WWW database access. In the future, we want to continue our experiments with various caching strategies and carry out detailed performance measurements.

Ultimately, all active caching on both the server and client sides involves the prefetching of data based on anticipated user requirements. Predicting user requests may depend both on individual user behaviour and also application characteristics. For example, semantic relationships between data may indicate that access to one data object is likely to be followed by access to another specific data object. We are therefore also investigating extensions to the user and application profiles and their construction to assist in prefetching strategies.

7 BIOGRAPHY

Antonia Erni is a Research Assistant in the Institute for Information Systems at ETH Zurich working in the area of agents and internet databases. She designed a generic agent framework for the development of integrated, multimedia information services for internet databases and developed a generic World-Wide Web interface to the OMS object-oriented database system. Currently, she is developing an internet apartment reservation system using the general framework.

Moira C. Norrie is a Professor in the Institute for Information Systems at ETH Zurich. She formed a research group in the area of Global Information Systems in February 1996. The main interests of the group are in object-oriented data models and systems, internet databases and advanced database applications such as document management systems, product information systems and scientific data systems.

Adrian Kobler is a Research Assistant in the Institute for Information Systems at ETH Zurich. He designed and implemented a product information system based on a general product data model using the OMS/Java framework. OMS/Java can be regarded as an object-oriented database management system for the Java environment that supports the generic object data model OM. Currently, he is developing the product data model into a general meta data model.

REFERENCES

Bayer, D. A Learning Agent for Resource Discovery on the World Wide Web. Master's thesis, Dept of Computing Science, University of Aberdeen, Scotland, 1995.

Doorenbos, R. B, Etzioni, O. and Weld, D. S. A Scalable Comparison-Shopping Agent for the World Wide Web. In *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, California USA, February 1997.

Erni, A. and Norrie, M. C. Agent Based Internet Database Services. In *4th Doctoral Consortium CAISE'97*, Barcelona, Spain, June 1997.

Erni, A. and Norrie, M. C. Snownet: An agent-based internet tourist information service. In *Proceedings of the 4th International Conference on Information and Communication Technologies in Tourism, Edinburgh, Scotland*, Institute for Information Systems, ETH Zurich, January 1997. Springer-Verlag.

Honegger, F. PIA: Personal Internet Assistant. Semester thesis, Institute for Information Systems, ETH Zurich, 1997.

Lang, K. NewsWeeder: Learning to Filter Netnews. In *Proc. 12th Intl. Machine Learning Conference (ML95)*, San Francisco, USA, 1995. Morgan Kaufmann.

Maes, P. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7), July 1994.

Maes, P. Modeling Adaptive Autonomous Agents. *Artificial Life Journal*, 1(1 & 2), 1994.

Sun Microsystems. The Java Development Kit (JDK). http://java.sun.com/products/jdk/.

Norrie, M. C. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In *12th Intl. Conf. on Entity-Relationship Approach*, pages 390–401, Dallas, Texas, December 1993. Springer-Verlag, LNCS 823.

O2 Technology Inc., Versailles, France. *O2Web.*

Oracle Corporation, California 94065. U.S.A. *Oracle WebServer 2.0*, 1996.

Payne, T. R. and Edwards, P. Interface Agents that Learn: An Investigation of Learning Issues in a Mail Agent Interface, 1995.

Ruser, P. Agent-based product data information system. Master's thesis, Institute for Information Systems, ETH Zurich, 1997.

Shakes, J., Langheinrich, M. and Etzioni, O. Dynamic Reference Sifting:

A Case Study in the Homepage Domain. In *Proceedings of the 6th International World Wide Web Conference*, Santa Clara, California USA, April 1997.

Voorhees, E. M. Software Agents for Information Retrieval. In *Software Agents: Papers from the 1994 Spring Symposium*, Menlo Park, CA, USA, 1994. AAAI Press.

Wuergler, A. Object Model System: An Object Database Management System for the OM Data Model. Master's thesis, Institute for Information Systems, ETH, 1995.