World Wide Web Database Integration via Mobile Agents

Quang M. Trinh, Ken Barker, and Reda Alhajj

Advanced Database Systems and Applications (ADSA) Laboratory

Computer Science Department

University of Calgary

Calgary, Alberta, T2N 1N4, Canada

{qtrinh, barker, alhajj}@cpsc.ucalgary.ca

November 26, 2002

World Wide Web Database Integration via Mobile Agents

Abstract

This paper presents an architecture that makes use of dynamic deployment of mobile agents to support both integration of heterogeneous/homogeneous databases on the World Wide Web (or web) and data exchange between the architecture and existing applications. The system uses an integrated and global schema to utilize agents in remote locations and HyperText Markup Language (HTML - usable even with minimal bandwidth) as the client interface and to control agents behind the scene. Using HTML reduces overhead and therefore gives the architecture performance advantages over other existing mobile agent systems. Furthermore, the system also supports the addition and removal of component databases dynamically. The exchange language used by the system is XML (eXtensible Markup Language), a standard and extensible format with supports in almost all programming languages, which can be read and understood by almost any application. With rich document structures like XML, not only can data transported by the Agents be used over the web, but it can also be used for Business To Business (B2B) data exchange between existing and/or future applications. The proposed system is shown to be better than existing techniques when higher volumes of data are to be exchange such as is typical B2B applications. The improvements are consistently significant and are magnified substantially as the amount of data exchanged increases.

1 Introduction

The idea of using the web to provide access to the global information resources has been an attractive research area. Today, combining the web and Relational Database Management Systems (RDBMS) is a dominant technology used for storing, processing, and distributing data. In the past 20 years, many client/server-based data access and exchange techniques for RDBMSs have been developed. Such techniques include: Remote Method Invocation (RMI) [8], Common Object Request Broker Architecture (CORBA) [1] and XML-RPC [10]. However, with the advances in technologies and the exponential growth in the number of resources on the web, it is necessary to provide access to multiple heterogeneous information sources. This, consequently, adds complexity to the building of such systems. In this paper, we describe a database integration and data exchange technique over the web using mobile agents. In particular, the technique uses the web as a medium, and agents to locate and deliver data by exploring how to exploit a multi-agent system in a homogeneous/heterogeneous database system. Our approach minimizes overhead between the clients and the component database hosts and effectively utilize agents in remote locations.

The major contribution of this paper is a technique, based on the use of mobile agents, that can facilitate data integration between various data sources on the web while protecting the data sources' autonomy while providing the user with a globally consistent data view. This is done while ensuring that the addition and removal of component databases can occur dynamically and data exchange occurs while still supporting existing applications. The goals of our approach are as follows. First, we show that through the use of metadata, mobile agents can easily and effectively provide access to multiple databases on the web. Second, we develop

a technique whereby component databases can be added or removed dynamically, while creating a flexible system architecture. Third, we make use of the XML [9], a markup language that has been widely used as an *ad-hoc* metadata standard. XML is a good choice because it supports enriched document structures and formats data results in support of B2B data exchange. The rest of this paper is organized as follows. Section 2 describes the agent-based architectural design framework and its major components. Section 3 describes the implementation detail of the framework, including detecting, capturing, integrating, and querying component databases using mobile agents. Section 4 provides performance evaluation and timing comparisons of the architectural framework and the traditional client/server model (Applets). Section 5 summarizes the work closely related to that presented here and Section 6 concludes the paper and provides directions for future research.

2 Architectural Design

The architecture is designed to explore the dynamic deployment of a mobile agent system to support querying and and integrate databases on the web. Figure 1 shows the overall architectural design of the WWW-DIMA (World Wide Web Database Integration via Mobile Agents) system with one client, three component databases (each with a DIMA Agent), and a central server (with the DIMA AgentManager where user queries are submitted). A typical scenario is as follows: using a web browser (Netscape or Internet Explorer), the client can pose a query to any of the component databases via the integrated and global schema on the central server.

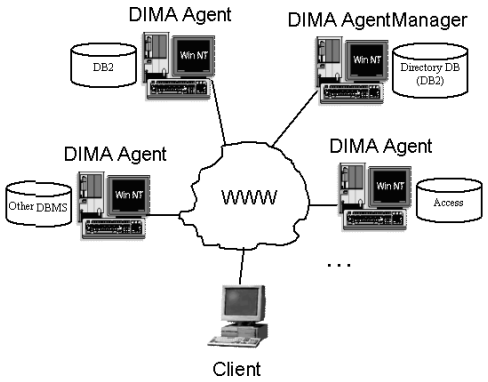

Figure 1: Overall architecture of the WWW-DIMA System.

The WWW-DIMA system consists of six independent components: the Directory Database, Agent Contexts, the DIMA ServiceAgentManager, DIMA ServiceAgents[1], the DIMA AgentManager, and DIMA Agents.

[1]Neither the DIMA ServiceAgentManager nor the DIMA ServiceAgents are shown in Figure 1 because they are transient components

The following summarizes the functionality of each component.

- The Directory Database contains the integrated global schema and the component database hosts information. The main purpose of this database is to maximize efficiency and utilize DIMA Agents in remote locations. Only the DIMA AgentManager and the DIMA ServiceAgentManager (both defined later) use the information in this database. When either the DIMA ServiceAgentManager or the DIMA AgentManager need to know the location of the requested data or the DIMA Agent who is responsible for the requested data, they can find this information in the Directory Database. This reduces the search time significantly and therefore increases the overall performance of the WWW-DIMA system. Information in this database includes: host names, host status (reachable or unreachable via the network), database instance names of all the component database hosts, *etc*.

- Agent Context is an application program that provides the resources and environment for agents to execute. Thus, an instance of the Agent Context is required for each component database host.

- The DIMA ServiceAgentManager coordinates and manages all the DIMA ServiceAgents in the system. Task assignment to individual DIMA ServiceAgent is based on the number of reachable hosts (and hence databases) in the Directory Database. DIMA ServiceAgents are periodically created and directed by the DIMA ServiceAgentManager. In particular, for each reachable component database host in the Directory Database, the DIMA ServiceAgentManager creates a DIMA ServiceAgent and assigns the database host to the newly created DIMA ServiceAgent. These DIMA ServiceAgents are responsible for checking and updating the status of the component database hosts as well as changes (if any) of the component database schemas. Any changes of the component schemas are detected by the DIMA ServiceAgent and are updated in the Directory Database on the central server by the DIMA ServiceAgentManager.

- The DIMA AgentManager is responsible for handling user queries from the web server and delivering query results from DIMA Agents (in the remote component database hosts) back to the central server. When a query is received, the DIMA AgentManager "decodes" the query into sub-queries (if necessary) and sends these sub-queries to DIMA Agents who are responsible for these sub-queries. By using the information in the Directory Database, the DIMA AgentManager can effectively determine the location of the requested data and the DIMA Agent who is responsible for the requested data. In the case where a component database host is unreachable, the DIMA AgentManager could, conceivably, also direct DIMA Agents to alternative component database hosts[2].

of the architecture that are created and activated as required.

[2]The current implementation of the WWW-DIMA system does not support this functionality.

- DIMA Agents are created and sent to component database hosts by the DIMA AgentManager. Similar to DIMA ServiceAgents, the number of DIMA Agents created depends on the number of reachable component database hosts in the Directory Database. Once created, each DIMA Agent is dispatched to the assigned remote (component database) host and waits for data request from the DIMA AgentManager. When a data request is received, the DIMA Agent responsible for the request will carry out the request and respond back with the query result.

3 Implementation Details of the WWW-DIMA System

There are four major parts in the implementation of the WWW-DIMA system: the detecting process, the capturing process, the integrating and the querying process. Figure 2 shows the implementation details of the WWW-DIMA system and how components within the WWW-DIMA system interact with each other.

3.1 Detecting Process

The detection process involves only the DIMA ServiceAgentManager and the DIMA ServiceAgent(s). The purpose of this process is to check and update the status and schema changes (if any) of component database hosts and component databases. The DIMA ServiceAgentManager creates and also tracks the number of DIMA ServiceAgents created so that it knows whether or not the DIMA ServiceAgents have successufully completed their assigned tasks. Periodically, the DIMA ServiceAgentManager sends out messages to the DIMA ServiceAgents requesting status reports and schema changes of the component hosts and component databases. If a DIMA ServiceAgent is "unreachable" (i.e., the DIMA ServiceAgent cannot be contacted via the network and therefore did not get the requesting message from the DIMA ServiceAgentManager), the DIMA ServiceAgentManager will then update the status (of the component database host the "unreachable" DIMA ServiceAgent is responsible for) as "unreachable" in the Directory Database. For other DIMA ServiceAgents (i.e. "reachable" DIMA ServiceAgents), reporting status and the schema changes are sent to the DIMA ServiceAgentManager on the central server and the changes are updated in the Directory Database.

In the component database hosts, there are two possible scenarios:

- Scenario 1: The central server is "unreachable" (i.e. the component database host is intentional/ unintentional disconnected from the network some time after the last detecting/updating check-point). This means the DIMA ServiceAgent on this host is disconnected from the network and hence did-not receive any of the reporting messages from the DIMA ServiceAgentManager. In this case, this component database host is considered "unreachable" in the Directory Database on the central server.

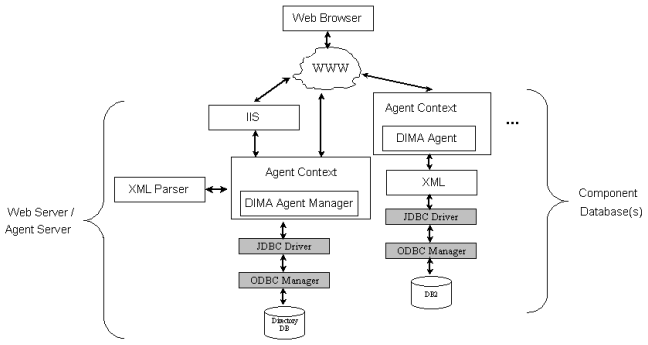

Figure 2: Implementation details of the WWW-DIMA System.

- Scenario 2: The central server is "reachable". In this case, when the requesting message from the DIMA ServiceAgentManager is received, the DIMA ServiceAgent extracts the meta-data changes (if any) and sends the changes back to the DIMA ServiceAgentManager on the central server.

3.2 Capturing Process

The capturing process requires that each DIMA ServiceAgent connects to the component database locally and extract the corresponding schema. The local naming convention used in our system is the name assigned locally at the component database. This means that different component databases can have the same local name and that they represent the same meaning. Thus, by combining the component database host name and the local name, we can uniquely identify any relation and/or any field from any of the component databases in the system. Figure 3 shows two vendor databases and their corresponding schemas. After extracted, the schema information, it is formatted into XML before it is composed into a message and sent to the DIMA ServiceAgentManager on the central server for integration. We chose XML as the exchange language between agents in our system (and hence with other legacy systems) for two reasons: (1) XML is an exchange language that describes both data and data structure, and (2) XML is known for its inherent extensibility and is well supported by most popular programming languages and environments.

Summary

Currently, the WWW-DIMA system uses a 5 minute time-frame to activate the detecting/capturing process. A shorter time-frame could generate unnecessary overhead and longer time-frame may result in higher chance of inconsistency between the integrated global schema and the component schemas. However, this time-frame can be adjusted according to the application needs. Each time it is activated, the DIMA AgentManager is also responsible for detecting whether if any of the "unreachable" database hosts from the last check-point

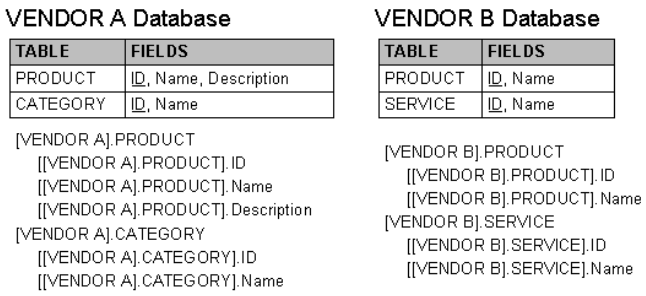

Figure 3: Two vendors databases and their corresponding schemas.

are now "reachable". If any one of the "unreachable" hosts is "reachable", then the capturing and updating process begins automatically. Once the changes are integrated into the Directory Database on the central server, the host status of this host is changed to "reachable" by the DIMA AgentManager. On the other hand, if none of these hosts are "reachable", then no changes are made in the Directory Database so they remain "unreachable".

3.3 Integration Process

The integration process integrates component schemas into an integrated and global schema on the central server. For each schema sent by the DIMA ServiceAgent, the DIMA ServiceAgentManager will either:

a) creates new mappings for a schema that is not already in the integrated schema.

b) discards mappings of the new schema that is already in the integrated schema where mappings of the new schema and mappings in the integrated schema are the same.

c) updates mappings in the integrated schema if the new schema is already in the integrated schema. This occurs when some mappings of the new schema have changed.

Since our naming convention is unique, naming conflicts between component database schemas are eliminated. Figure 4 shows the integrated schema of the two vendor databases and their corresponding mappings.

3.4 Querying Process

Users pose queries through a browser directly to the integrated and global schema and get their query results back from the DIMA AgentManager. All queries submitted and their results are in the form of HTTP POST and HTTP RESPONSE. For each query (or request) submitted by the user, the DIMA AgentManager

Integrated View	Corresponding Mapping
PRODUCT	[Vendor A].PRODUCT [Vendor B].PRODUCT
ID	[Vendor A].PRODUCT.ID [Vendor B].PRODUCT.ID
NAME	[Vendor A].PRODUCT.NAME [Vendor B].PRODUCT.NAME
DESCRIPTION	[Vendor A].PRODUCT.DESCRIPTION
CATEGORY	[Vendor A].CATEGORY
ID	[Vendor A].CATEGORY.ID
NAME	[Vendor A].CATEGORY.NAME
SERVICE	[Vendor B].SERVICE
ID	[Vendor B].SERVICE.ID
NAME	[Vendor B].SERVICE.NAME

Figure 4: Integrated schema of the two vendor databases and their corresonding mappings.

decodes[3] the query using the information in the Directory Database and forwards the request to the DIMA Agent who is responsible for the request. The DIMA AgentManager then waits for the query result from the DIMA Agent handling the request in the remote component database host. For queries that required data from multiple locations (and hence multiple DIMA Agents), the DIMA AgentManager is also responsible for de-composing the query into sub-queries and combining the sub-query results from multiple DIMA Agents.

When the request message is received on the component database host, the DIMA Agent connects to the component database locally and extracts the requested data. The extracted data or the query result is then formatted into XML before it is sent back to the DIMA AgentManager on the central server. An advantage of formating the query result in XML is that this XML-query result can be further processed by other intermediate applications that can understand XML. Furthermore, this feature allows the WWW-DIMA system to integrate and exchange data easily with other third-party applications while maintaining the WWW-DIMA system's properties.

When the XML-query result is received on the central server, the DIMA AgentManager parses the query result and formats it into a HTML page. This includes generating appropriate heading and HTML tags for the HTML result page. When the formatting is done, this HTML page is forwarded to the web server, which is then presented to the user.

3.4.1 Query Example and Query De-composition

The following two examples show how queries posed to the integrated schema are de-composed and mapped to component databases.

Example 1: In this example, we assumed that the user is interested in a listing of all available product names. Consider the submission of a user query such as: SELECT PRODUCT.NAME FROM PRODUCT.

[3]Decoding is the process of determining the location of the requested data and the the DIMA Agent who is responsible for the request.

Since there are two mappings for PRODUCT.NAME in the integrated schema (Figure 4), the query is decomposed into two sub-queries by the DIMA AgentManager and assigned to the two DIMA Agents on the two component database hosts. The two sub-queries generated by the DIMA AgentManager from the original query are:

```
For vendor A
        SELECT [VENDOR A].PRODUCT.NAME
        FROM [VENDOR A].PRODUCT

For vendor B
        SELECT [VENDOR B].PRODUCT.NAME
        FROM [VENDOR B].PRODUCT
```

Each sub-query is then executed by the DIMA Agent at the component database host and the results are sent back to the DIMA AgentManager on the central server where they are combined. Both the mappings and the sub-query results combination are done by the DIMA AgentManager so they are transparent to the users.

Example 2: In this example, we assumed that the user is interested in a listing of all available service names in the system. When the user poses the query (SELECT SERVICE.NAME FROM SERVICE) to the integrated schema, there is exactly one mapping for this query (hence one location and one DIMA Agent) so no result combining is required. The equivalent mapping of the original query is:

```
For vendor B
        SELECT [VENDOR B].SERVICE.NAME
        FROM [VENDOR B].SERVICE
```

4 Performance Evaluation

To simulate a heterogeneous environment, different DBMSs were used: Microsoft Access and IBM DB2. Furthermore, we also need to compare the performance of our system with other well-know approaches such as Applets or Servlets. Like the WWW-DIMA system, both Applet and Servlet use an HTTP server to connect to the database, so they represent a fair comparison. For our experiments, we chose Applets because past work exits using Applets as the client interface and we wanted to compare their performance with our system.

The same queries were evaluated using both the Applet implementation and the WWW-DIMA implementation. The evaluation for both implementations was performed on a network of workstations (Pentium

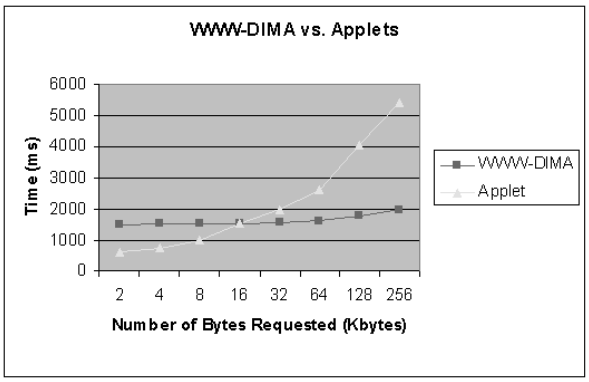

Figure 5: Comparison of the WWW-DIMA Implementation and Applet Implementation during off-peak hours.

4, 1.4GHz processor speed with 512 MB of RAM) running Windows 2000 on 100Mbps switched Ethernet. The timing observations for the two implementations were done at two different times: at peak hours (when the network traffic was high) and during off-peak hours (when the network traffic was low). The timings for both implementations were repeatedly observed and obtained. To ensure variation, the size of the data results requested were set to vary from 2 Kbytes to 256 Kbytes to reflect a typical B2B environment.

Figure 5 shows a comparison of the WWW-DIMA implementation and the Applet implementation during off-peak hours. Note that the performance of Applets is better than that of the WWW-DIMA system when the requested data is between 2 Kbytes and 16 Kbytes. This is because of the overhead of the agent framework (Aglets) [5] used by the WWW-DIMA system. Recent studies of Aglets [3, 7] have suggested that the main overhead of Aglets is the Agent Transport Protocol (ATP). The overhead of the ATP is relatively large compared to the actual transfer time of the requested data (small). Since the WWW-DIMA implementation uses the Aglet's ATP, its performance is substantially affected because of the additional overhead. Experiments with the WWW-DIMA system show that as the size of the requested data increases, the total query time (i.e. the time from when the request is submitted to the time when the query result is presented) increases only slightly. When the size of the requested data exceeds 16 Kbytes (up to and including 256 Kbytes) performance of the WWW-DIMA system is better than that of Applets. Typically, the amount of data exchange in any B2B environment is large and with the improvements consistently shown from our experiment (Figure 5), we can conclude that the WWW-DIMA systems is appropriate to support data exchange in any B2B environment.

The timing comparison of the WWW-DIMA system and Applet implementation during peak hours is very similar to that of off-peak hours (Figure 5). In fact, the apparent differences between peak hours and off-peak hours are negligible. There are two reasons for this. First, the overhead of the ATP is large compared to the additional overhead due to higher network traffic. Second, our experiments were done on a 100Mbps

switched Ethernet so there are very minor performance changes between peak hours and off-peak hours.

5 Related Work

Many working prototypes of systems that support distributed information retrieval using mobile agents have been implemented and evaluated recently. Although, all the prototypes make use of the same software engineering paradigm, mobile agents, their approaches are different and used varying agent frameworks. We discuss some key contributions here.

Papastavrou *et al.* [7] developed a very similar model to that of the WWW-DIMA system. Unlike the WWW-DIMA prototype, however, Papastavrou *et al.* use Applets as the client interface and to control the agents behind the scene. An advantage of using Applets as an interface is that other standard Java GUI components can easily be integrated. However, the more components that are added to the Applet, the slower the download of the Applet. Papastavrou *et al.* handle all client queries with three different techniques. The first technique is, for every query submitted by the client, Papastavrou *et al.* create and dispatch an agent (with client's query) from the client to the web-server, then to the location of data. Upon arrival at the location of data, the agent establishes a local connection to the data source and extracts the requested data. The agent then dispatches itself back to the client with the requested data. The major overhead of this technique is the migration of the agent for every single query submitted (from the client to the web server, to the location of data, and back to the client). In the second technique, Papastavrou *et al.* use two agents: a messenger agent and a database agent. Both agents are dispatched when the first client query is submitted. The messenger agent is responsible for carrying the request from the client to the location of data and the requested data back to the client. The database agent, on the other hand, is dispatched to the location of data from the client when the first query is submitted. Unlike the messenger agent, the database agent resides at the location of the data source for the rest of the application's execution. The database agent is responsible for establishing a local database connection and extracting the requested data. The design of this technique is better than the first technique, however, the performance of this technique is not significantly different. In the third technique, Papastavrou *et al.* use messages. That is, when a query is submitted by the client, a message is sent from the client to the database agent at the location of data. When the database agent receives the message, it extracts the requested data, and sends the data result back to the client via another message. The performance of this technique is significantly improved compared to the first two techniques. Furthermore, Papastavrou *et al.* have also used the first technique in querying multiple data sources. Papastavrou *et al.*'s performance evaluation results confirm that their three techniques outperforms the conventional client/server model, Applets, and that using messages (the third technique) has the best performance. In particularly, on

a fixed network, the performance gains of their third technique was between 30% and 40%. In comparison with the WWW-DIMA system, both the WWW-DIMA system and Papastavrou *et al.*, in particular using the third technique, are very similar models except that Papastavrou *et al.* use Applets as the client interface and the WWW-DIMA system uses HTML. Since Applets require download time and HTML does not, the WWW-DIMA has performance advantage over Papastavrou *et al.*

Ismail *et al.* [3] compared the performance of RMI (Remote Method Invocation), Aglets and a minimal self-built mobile agent prototype. All three models were built using Java. Ismail *et al.* consider two application scenarios: the *forward* application and the *compress* application. In the forward application, the client must sequentially connect to two data sources to collect information. In the compress application, client connects directly to the server when requesting a document. On the server side, the requested document is extracted, compressed and then sent back to the client. The timing performance evaluations were done in both applications using the three models (RMI, Aglets, and self-built mobile agent prototype). In the forward application, RMI uses two Remote Procedure Calls (RPCs) sequentially to connect to the two data sources. For both mobile agent prototypes (Aglets and the self-built mobile agent), agents are created from the client side, sequentially dispatched to the first data source, then the second data source, and then back to the client with the requested data. In the forward application, Ismail *et al.*'s timing performance shows that for smaller sizes (approximately less than 2400 bytes) of the requested data, RMI is better. For larger data sizes (approximately greater than 2400 bytes), both mobile agent prototypes have better performance than that of RMI. In the compress application, RMI's performance is better than both of the agent prototypes when the size of the document is less than 100 Kbytes. For all documents between 100 Kbytes and 2,000 Kbytes, RMI's performance is better than that of Aglets, and worse compared to the self-built agent prototype. Ismail *et al.*'s self-built agents outperforms Aglets in both applications. This is due to the significant overhead of the Aglets' Agent Transfer Protocol (ATP). Although, performance comparison between the WWW-DIMA system and Ismail *et al.*'s is inappropriate (because Ismail *et al.* sequentially collect information from two data sources and therefore has much more additional overhead compared to the WWW-DIMA system), however, the WWW-DIMA's technique is better and more efficient (confirmed by [7]) compared to Ismail *et al.*'s. That is, the WWW-DIMA system uses HTML for client interface and message passing for requesting data.

Gray *et al.* [2] compared the performances of four different agent-based frameworks (D'Agents, EMAA, KAoS, and NOMADS) with the traditional client/server technique, Remote Procedure Call (RPC), in distributed information retrieval. In the agent-based approach, agents are dispatched from the client hosts to a server where a collection of documents is stored. On the server, the agents search for the requested document(s), filter out unwanted documents and return only the requested document(s). In the client/server approach (written in C++), clients connect to the server, download all available documents, and filter for the requested document(s) on the client-side. Though, the four frameworks share a common implementation

language, Java, their performance is very different. These differences are due to the internal implementation of each framework. (For example, both EMAA and KAoS cache the open socket on the agent server and re-use the socket. This saves the overhead when migrating agents to the server. Caching and re-using these sockets saves overhead in the overall performance. D'Agents and NOMADS, on the other hand, open a new socket for each migration, which results in additional overhead). Gray *et al*. conclude that first, mobile agents can be beneficial in low network bandwidth, though increases server load. Second, migration time is a major factor of the overall performance of mobile agent models (i.e.: migration time depends largely on the network bandwidth). Third, client/server models are beneficial when the network bandwidth is large. Compared with the WWW-DIMA system, Gray *et al*., like many others, migrate agents from clients to the server and therefore has much more additional and unnecessary overhead.

6 Conclusion

Past research has shown that using the traditional client/server model for accessing multiple data sources on the web is often slow and can be complicated to implement. In addition, the client/server model changes the performance of the applications and the network bandwidth as the number of users increase. The WWW-DIMA architecture and implementation described in this paper is both flexible and scalable. The architecture of the WWW-DIMA is flexible because the deployments of DIMA Agents are done dynamically and therefore reduces the overall complexity of the system. Further, the WWW-DIMA architecture periodically uses DIMA ServiceAgents to check and update status and schema changes of component data sources. The WWW-DIMA architecture is scalable because additional data sources can be added or removed dynamically. With the use of mobile agents (DIMA Agents), the WWW-DIMA system allows users to effectively access multiple data sources on the web and yet manages the complexity of the system.

Other performance advantages of the WWW-DIMA system compared to other recent implementations include the following. First, unlike other work, the WWW-DIMA implementation uses message passing instead of agent dispatching. Dispatching agents can be costly as the transfer time is directly related to the total number of bytes transferred across the network. That is, in addition to the requested data, internal state and code of agents are transferred over the network as well. Second, the WWW-DIMA implementation uses HTML as the client interface thus usable even with minimal bandwidth. Third, data exchanged on the web occurs while supporting existing applications (i.e. data exchange between DIMA Agents are presented in XML, a format that is well supported by many programming languages and environments and can be read and understood by almost any application).

Aside from handling concurrent multiple requests, other future work for the WWW-DIMA system include:

- Support for alternate data sources: as discussed in Section 2, in the case where a data source is unavailable, the DIMA AgentManager can still redirect the request to alternative data source(s).
- Integrate with other existing client applications: with the use of XML, other existing client applications can integrate and exchange data easily with the WWW-DIMA system.
- Support for disconnected operations: mobile agents are appealing to wireless clients [4] because mobile agents can accept requests, disconnect from their clients, move to location(s) of data to be accessed, carry out requested tasks and bring back results when their clients are reconnected.

References

[1] Object Management Group (OMG), *CORBA/IIOP Specification*. Technical Report, June, 2002

[2] R. S. Gray, D. Kotz, R. A. Peterson Jr., J. Barton, D. Chacon, P. Gerken, M. Hofmann, J. Bradshaw, M. Breedy, R. Jeffers, and Ni. Suri. *Mobile-Agent versus Client/Server Performance: Scalability in an Information-Retrieval Task*. In the 5th International Conference on Mobile Agents (ICMA), p. 229-243, Atlanta, December, 2001

[3] L. Ismail and D. Hagimont: *A Performance Evaluation of the Mobile Agent Paradigm*. In the 14th Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA), p. 306-313, Denver, November, 1999

[4] D. Kotz, G. Jiang, R. Gray, G. Cybenko, R. A. Peterson. *Performance Analysis of Mobile Agents for Filtering Data Streams on Wireless Networks*. In Proc. of the 3rd ACM International Workshop on Modeling and Simulation of Wireless and Mobile Systems, p. 85-94, Boston, August, 2000

[5] D. B. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addition Wesley, Reading, MA, USA, 1998

[6] M. Magnanelli and M. Norrie. *Databases for Agents and Agents for Databases*. In Proc. of the 2nd International Bi-Conference Workshop on Agent-Oriented Information Systems, June, 2000

[7] S. Papastavrou, G. Samaras, and E. Pitoura, *Mobile Agents for WWW Distributed Database Access*. In the 15th International Conference on Data Engineering (ICDE99), p. 228-237, Sydney, 1999

[8] A. Wollrath, R. Riggs, and J. Waldo. *A Distributed Object Model for the Java System*. In Proc. of USENIX Conference on Object-Oriented Technologies (COOTS), p. 219-231, Toronto, June, 1996

[9] Extensible Markup Language (XML) 1.0 (Second Edition), The World Wide Web Consortium, available at http://www.w3.org/TR/REC-xml

[10] XML-RPC Specification, UserLand Software, Inc., available at http://www.xmlrpc.com/spec