Mobile Agents for World Wide Web Distributed Database Access

Stavros Papastavrou, *Student Member*, *IEEE*,
George Samaras, *Senior Member*, *IEEE*, and Evaggelia Pitoura, *Member*, *IEEE*

Abstract—The popularity of the Web as a universal access mechanism for network information has created the need for developing web-based DBMS client/server applications. However, the current commercial applet-based approaches for accessing database systems offer limited flexibility, scalability, and robustness. In this paper, we propose a new framework for Web-based distributed access to database systems based on Java-based mobile agents. The framework supports lightweight, portable, and autonomous clients as well as operation on slow or expensive networks. The implementation of the framework using the aglet workbench shows that its performance is comparable to, and in some case outperforms, the current approach. In fact, in wireless and dial-up environments and for average size transactions, a client/agent/server adaptation of the framework provides a performance improvement of approximately a factor of ten. For the fixed network, the gains are about 40 percent and 30 percent, respectively. We expect our framework to perform even better when deployed using different implementation platforms as indicated by our preliminary results from an implementation based on Voyager.

Index Terms—Mobile agents, aglet, distributed computing, JDBC, web data access, DBMS-aglet, mobile computing.

1 Introduction

The widespread use of Java [1] in network-centric computing, attributed mainly to its portability and security control system, gives Java the lead in client/server programming and mobile computing [2]. Moreover, the already established Java database connectivity application interface (JDBC API) [3], [4] and the constantly growing and refining of JDBC [5] drivers have drawn the attention of major database vendors. On the other hand, the World Wide Web (simply Web) [10], [11], [12] is rapidly being accepted as a universal access mechanism for network information. The popularity of the Web suggests that Web browsers may offer a compelling end-user interface for a large class of applications, including database management systems (DBMSs).

Thus, an important issue is to combine these two technologies, namely Java and Web, for the retrieval of information residing in database systems. The real challenge is the formation of smart, lightweight, flexible, independent, and portable Java DBMS client programs that will support database connectivity over the Internet. However, the currently proposed approaches [5] (i.e., the applet/JDBC based ones) overload the client, in terms of the size of downloaded code, and offer limited flexibility and scalability. In this paper, we introduce a new approach for the development of Java-based distributed client/server applications over the Web. Our approach is based on using mobile agents [6], between the client program and the server machine, to provide database connectivity, processing and communication, and consequently eliminate the overheads of the existing approaches.

The proposed framework, called the "DBMS-Aglet Framework," utilizes the technology of mobile agents and demonstrates its effectiveness over a specific application context (i.e., DBMS access). The framework is comprised of a set of Java based agents that cooperate to efficiently support Web database connectivity. The main agent, called DBMS-aglet, acquires its database capabilities dynamically, not at the client but at the server. The other agents of the framework assist this dynamic acquisition. This idea promotes a much more efficient way of utilizing the JDBC API and the JDBC driver API and eliminates the overheads of the various conventional approaches. Consequently, it frees the remote client to perform other tasks.

The new form of Web-based database access supported by "DBMS-Aglet Framework" is shown to be more flexible, scalable, and robust than the current JDBC-based database connectivity. Furthermore, the framework supports lightweight, portable, and autonomous clients as well as operation on slow or expensive networks. The framework is generic and portable and can be used not only within the Web but stand-alone as well for direct Java database connectivity. Although, in this paper, we present performance results for accessing relational databases, it is worth pointing out that our approach is not restricted to relational databases but it can be used to access any type of database or file system. This is an additional strong point of the "DBMS-Aglet Framework," since it allows accessing different types of resources in a seamless manner.

The implementation of the framework shows that its performance is comparable to, and in some cases outperforms, the performance of current approaches. In fact, in

- *S. Papastavrou and G. Samaras are with the Department of Computer Science, University of Cyprus, CY-1678 Nicosia, Cyprus. E-mail: cssamara@cs.ucy.ac.cy.*
- *E. Pitoura is with the Department of Computer Science, University of Ioannina, Greece. E-mail: pitoura@cs.uoi.gr.*

Manuscript received 10 Sept. 1999; revised 10 July 2000; accepted 10 July 2000.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 112702.

wireless and dial-up environments and for average size transactions, a certain adaptation of the framework provides a performance improvement of approximately a factor of ten. For the fixed network, the gains are about 40 percent and 30 percent, respectively. These performance results were gained while using the Aglets Workbench [7] for the implementation of mobile agents. Since the Aglets Workbench is more tuned towards functionality than performance, we expect our framework to perform even better. This assumption is substantiated by early experiments conducted using Voyager [22] as our implementation platform. We report these experimental results as well.

The remaining sections of this paper are organized as follows: Section 2 presents the needed background material. This includes a short introduction to Java, mobile agents, and mobile aglets along with an evaluation of the current applet-based approaches for web access to distributed databases. In Section 3, we present the proposed DBMS-aglet framework, discuss its adaptations and effectiveness within the various client/server computational models, and present and compare its advantages to the current approaches. A variation of the framework and its advantages for accessing distributed multidatabase systems is discussed in Section 4. In Section 5, we present a Java-based hierarchy and generalization of the various frameworks and discuss future work. A specialized-agents library for database services is also presented. Performance evaluation is presented in Section 6. Implementation issues and the role of the implementation platform are discussed in Section 7. Section 8 concludes the paper.

2 Background Material

2.1 Supporting Technologies

2.1.1 Java and JDBC

Java is an object-oriented, interpreted, robust, secure, architecture neutral, portable, and multithreaded language. The uniqueness of Java lies on the fact that it combines both compiled and interpreted code. The Java executable code (called bytecode) runs on any hardware platform with a Java interpreter or any Java-enabled Web browser. In particular, the Java bytecode represents the instructions for a virtual microprocessor called the Java Virtual Machine (JVM). The Java Virtual Machine, also known as the "Java Interpreter" [2], is an abstract computer that runs Java compiled programs. The JVM is "abstract" in the sense that it is software based and runs over various hardware platforms. Another key characteristic of Java is the small size of its compiled code. This feature enables Java compiled classes to travel efficiently through the Web, making it very attractive for network-centric programming. A Java applet is a Java object-program that can run within the context of a Java enabled web browser. A downloaded Java applet can perform tasks only within the context of the client's hosting web browser and it is not allowed to access any local resources of the client for security considerations. Another restriction is that Java applets are not allowed to communicate with URLs other than the one they were downloaded from.

The Java Database Connectivity (JDBC) is the Java standard specification for accessing and manipulating relational databases [4]. The JDBC consists of two layers: The JDBC API that provides a Java interface to the relational database, and the JDBC driver API that executes/implements this interface. A client that employs the JDBC API must first download to its environment a JDBC driver before accessing a particular database. In this work, the version of the JDBC drivers used was 1.1.

2.1.2 Mobile Agents

Mobile agents are processes dispatched from a source computer to accomplish a specified task [14], [15]. After its submission, the mobile agent proceeds autonomously and independently of the sending client. When the agent reaches a server, it is delivered to an agent execution environment. Then, if the agent possesses necessary authentication credentials, its executable parts are started. To accomplish its task, the mobile agent can transport itself to another server, spawn new agents, or interact with other agents. Upon completion, the mobile agent delivers the results to the sending client or to another server.

Aglet Technology [7] (also known as the Aglets Workbench) is a framework for programming mobile network agents in Java developed by the IBM Japan research group. The IBM's mobile agent, called "Aglet" (agile applet), is a lightweight Java object that can move autonomously from one computer host to another for execution, carrying along its program code and state as well as the data so far obtained. One of the main differences between an aglet and the simple mobile code of Java applets is the itinerary or travel plan that is carried along with the aglet. By having a travel plan, aglets are capable of roaming the Internet collecting information from many places. The itinerary can change dynamically.

An aglet can be dispatched to any remote host that supports the Java Virtual Machine. This requires from the remote host to preinstall Tahiti, a tiny aglet server program implemented in Java and provided by the Aglet Framework. A running Tahiti server listens to the host's ports for incoming aglets, captures them, and provides them with an aglet context (i.e., an agent execution environment) in which they can run their code from the state that it was halted before they were dispatched. Within its context, an aglet can communicate with other aglets, collect local information and, when convenient, halt its execution and be dispatched to another host. An aglet can also be cloned or disposed.

To allow aglets to be fired from within applets, an abstract applet class, called "FijiApplet," is provided as part of a Java package, called "Fiji Kit." The FijiApplet maintains some kind of an aglet context (like the Tahiti aglet server). From within this context, aglets can be created, dispatched, and retracted back to the FijiApplet. For a Java-enabled web browser to host and fire aglets to various destinations, two additional components are provided by the Aglet Framework. These are an aglet plug-in that allows the browser to host aglets and an aglet router that must be installed at the Web server to capture incoming aglets and forward them to their destination.

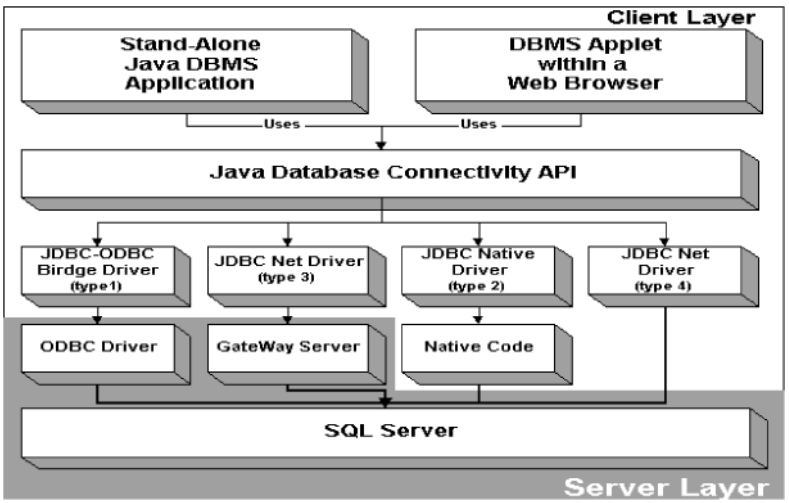

Fig. 1. Standard JDBC approaches.

2.2 The Current Approach: Applets for Distributed Database Access

The challenge is to provide the Internet user with a Web-based database connectivity with the lowest possible overhead. To this end, various non-Java-based approaches (e.g., CGI) have been proposed [16], [17]. The concept, however, of running applets within Java enabled browsers that utilize the JDBC application interface is increasingly gaining popularity with major database vendors. There are four ways a Java applet can gain access to a remote relational database (Fig. 1), making use of one of the four existing types of JDBC driver implementations.

The first approach makes use of the JDBC-ODBC bridge driver (JDBC driver type 1). The JDBC-ODBC bridge translates the JDBC API calls into ODBC[1] calls and sends them to an ODBC driver already installed on the server. This approach requires the client applet to download, along with its code, some ODBC binary code. Moreover, in many cases, client database code must be preinstalled on the client machine. Thus, this approach poses many extra layers of overhead. Nevertheless, it is useful in accessing databases that do not directly support JDBC [5].

The second approach makes use of a JDBC driver written half in Java and half in native[2] code. The client applet, through this JDBC driver (type 2 driver), speaks directly to the protocol of the remote SQL database. While this approach is very efficient in terms of performance, it requires the preinstallation of native code at the client.

The third approach is considered to be the most flexible: It uses a JDBC driver written entirely in Java meaning that the entire driver can be downloaded to the client applet. The client applet, through this JDBC driver (type 3 driver), speaks an intermediate language that is translated by a middle-tier gateway at the server into a DBMS-specific protocol and eventually passed to the SQL server. The more vendor protocols the gateway speaks, the more databases the applet can be attached to simultaneously. Despite the extra layer of the gateway, this approach has drawn the attention of many database vendors, including Borland with DataGateway [18], IBM with DB2 client support for Java [19], and Symantec with dbANYWHERE [20].

Finally, the fourth approach makes use of a JDBC driver (type 4 driver), also written entirely in Java. The driver can be fully downloaded to the client applet and speaks a DBMS-vendor protocol directly to the remote SQL server. The approach can efficiently serve the Internet user over the Web, but for attaching to various SQL servers, several JDBC drivers need to be loaded. Borland and Sybase have already released type 4 JDBC drivers.

All four existing approaches require, to some extent, downloading and initiating the JDBC driver on the client machine, which is generally a very resource-consuming procedure. Our primary concern is to simplify and relieve the remote client so that it does not need to handle a complex set of JDBC interface classes, but just the input of requests and the expected formatting of the output.

3 DBMS-Aglets: Mobile Agents for Distributed Database Access

In a nutshell, our idea is to use mobile agents between the client and the server machine. Instead of having a DBMS-applet at the client machine that downloads from the remote SQL server, initiates a JDBC driver, and handles a complex set of JDBC interfaces, our proposed DBMS-applet creates and fires a mobile agent (or agents if necessary) that travels directly to the remote SQL server. At the SQL server, the mobile agent initiates a local JDBC driver, connects to the database and performs any queries specified by the sending client. When the mobile agent completes its task at the SQL server, it dispatches itself back to the client machine directly into the DBMS-applet from where it was initially created and fired. Since our mobile agents posses database capabilities, they are called ***DBMS-aglets***.

3.1 The DBMS-Aglet Framework

To realize our approach, a number of processes are defined to complement the existing agent execution environment (i.e., the aglets). In particular, applets need to be enhanced to provide database specific interfaces and be capable to

1. Open Database Connectivity: An API that defines the routines for accessing MS-windows databases.
2. Native code: Database-vendor specific code.

TABLE 1
The Web-Based DBMS-Aglet Infrastructure

Standard Infrastructure	The Aglet Framework	The Dbms Aglet Framework
A web server	A Tahiti aglet server at the SQL server	A DBMS-applet
A Java-enabled browser	An aglet router installed at the Web server machine	A DBMS mobile agent (the DBMS-aglet)
An SQL server	An aglet plug-in for Java-enabled browsers to enable a web browser to host aglets	A DBMS-assistant stationary aglet

host agents with database capabilities. In addition, the existing aglets need to be extended to be database capable. Finally, supporting aglets need to be provided to assist such database-capable aglets in their negotiations with the SQL server.

Specifically, to support the DBMS aglet framework, the following components are needed:

- *A DBMS-applet*: The DBMS-applet is responsible for forming a graphical client database interface that the user can utilize to input database requests. Our suggested DBMS client applet is an extension of the abstract FijiApplet class.
- *A DBMS-aglet*: The DBMS-aglet is created within the context of the DBMS-applet and is responsible for carrying the user's request directly to the remote database, executing it, and returning the results back to the DBMS-applet context. Our suggested DBMS-aglet is a Java-based extension of the Aglet class.
- *A DBMS-assistant stationary aglet*: The DBMS-assistant stationary aglet resides at the site of the SQL server. Its responsibility is to inform any incoming DBMS-aglets carrying database requests about the available JDBC drivers and data sources and to assist them in carrying out their requests. Our suggested DBMS-assistant stationary aglet is an extension of the Aglet class.

The DBMS-applet, the DBMS-aglet, and the DBMS-assistant aglet compose the suggested "DBMS-Aglet Framework" that provides DBMS-capable mobile agents for distributed database access. The DBMS-Aglet Framework builds on and extends a mobile agent framework, in our case, the Aglets Workbench [7], that provides the facilities for hosting and routing mobile agents. In turn, the mobile agent framework can exploit a standard networking infrastructure, such as tools and services for Web-based access. Table 1 summarizes the complete Web-based DBMS-aglet infrastructure that consists of 1) the DBMS-aglets framework, 2) the aglets framework, and 3) the standard networking infrastructure.

3.2 Demonstrating the Web-based DBMS-Aglet Framework

To demonstrate our approach, we have set-up a Web-based DBMS-aglet infrastructure. To this end, we have programmed a DBMS-applet and a DBMS-aglet and included them in an html page at the web server machine. Additionally, we have installed an aglet router at the web server machine and a Tahiti aglet server at the SQL server machine. We have also developed a DBMS-assistant stationary aglet and initialized it within the Tahiti aglet context at the SQL server machine. Then, we downloaded, at the client host, the html page containing the DBMS-applet and the DBMS-aglet. Through the DBMS-applet's GUI, we entered a database request and order the DBMS-applet to carry it out. The DBMS-applet fired a DBMS-aglet that returned with the results.

Fig. 2 demonstrates the life-cycle of the DBMS-Aglet of the previous typical scenario. Execution begins with a web user downloading into his Java-enabled browser an html page that contains a DBMS-applet and the DBMS mobile agent (a DBMS-aglet) (Step 1).

After it has been initiated, the DBMS-applet forms a front-end DBMS client application interface. This can be achieved by making use of Java's standard GUI components (also known as AWT components [8]) to design an appropriate database interface (see, for example, [9]).

Once the database query is passed to the DBMS-applet interface, the DBMS-applet creates a lightweight DBMS mobile agent (the DBMS-aglet) within its context. The aglet then receives information from its creator DBMS-applet regarding the nature of the user's query. These directions must include among others:

- The address of the URL where the SQL server is located.
- The SQL query to be executed at the SQL server.
- The appropriate certificates for the aglet to be trusted at the SQL server.
- The route itinerary in case the agent is to visit more than one server.

Then, the DBMS-applet dispatches the aglet from its context to the URL of the SQL server (Step 2). Due to Java applet security restrictions mentioned earlier, the aglet must first go through the web server URL from where the html page containing the DBMS-aglet was downloaded. At the Web server machine, an aglet router captures the incoming DBMS-aglet and immediately forwards it to its destination (Step 3).

Arriving at the SQL server machine, the DBMS-aglet is received by a Tahiti aglet server. The DBMS-aglet communicates with the DBMS-assistant stationary aglet to be informed of the available JDBC drivers to load, and the available data sources to get connected to. Once this is done, and always within the Tahiti's context, the DBMS-aglet connects to the appropriate SQL server (the specific data

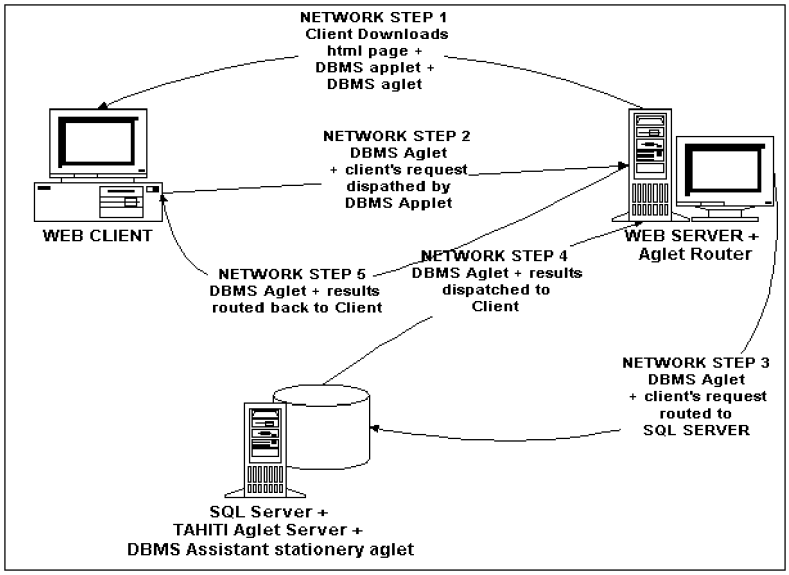

Fig. 2. The DBMD-Aglets life cycle.

source) to execute the query. When the aglet completes its task at the SQL server, it can either dispatch itself to another SQL server to perform another user's query or it can dispatch itself back to the DBMS-applet at the client machine, going through the aglet router at the web server machine (Step 4 and 5).

Arriving back to its original context, the aglet delivers the results to the DBMS-applet, which then presents them to the user through the graphical interface.

3.3 Refining the DBMS-Aglet Framework

The role of the aglet in the DBMS-aglet framework is to convey the various database requests to the SQL server and bring back the result. By simply replacing any direct requests with aglets that encapsulate and carry these requests, the remote client is relieved from the responsibility of downloading JDBC drivers. Downloading JDBC drivers and connecting to the database is now the responsibility of the various aglets. Unfortunately, though, this is done each and every time a request is issued and an aglet is fired to the database server, thus introducing an unnecessary and undesirable overhead. Figs. 7, 8, and 9 in Section 6 clearly confirm this overhead.

3.3.1 DBMS-Aglet and the Messenger Aglet

To eliminate this overhead, an extension of the client/server model, called the client/agent/server model (c/a/s) [21] is employed. In this three-tier architecture, an agent is placed on the path from the client to the server. Any communication between the client and the server goes through this agent. In our case, this agent is a service-specific (namely database connectivity) surrogate of the client which is placed (parked) at the SQL server and maintained there for the duration of the application. Between the parked aglet and the remote client, another aglet carries requests and results back and forth.

Based on this variation, upon the first client request, two DBMS-aglets are fired from the DBMS-applet. The first one is called parked DBMS-aglet. Its role is to "camp" at the SQL server's agent context, load the appropriate JDBC driver, connect to the database, submit the request, and collect and filter the answer. The second DBMS-aglet is called the messenger aglet. The messenger aglet is responsible for carrying the result back to the DBMS-applet (see Fig. 3). Any subsequent requests are transmitted via the messenger aglet to the parked DBMS-aglet.

This scheme is proved to be very efficient in cases where the user issues, through the DBMS-applet, a number of consecutive database requests to the same remote SQL servers. Furthermore, since these two aglets are (almost!) identical,[3] they own the same itinerary and, thus, if the parked DBMS-aglet moves to another server the messenger can deterministically follow it. An additional benefit of this approach is the ability of the messenger aglet to roam, if needed, around the net before returning to the client.

3.3.2 DBMS-Aglet Using Messages

The deployment of a parked DBMS-aglet attached locally to the database server eliminates one overhead, namely the time to reload the JDBC, and reconnect to the database for each and every query. The other overhead left is the time required for the messenger aglet to travel between the DBMS applet and the parked DBMS-aglet carrying results and new queries.

Replacing the messenger aglet with two messages can eliminate this overhead. The first message is delivered from the parked DBMS-aglet to the DBMS applet and contains the results of the last query. The other message from the DBMS applet to the DBMS-aglet contains the new client query and any additional directions to the parked DBMS-aglet that might be needed. This approach demonstrates a true service-specific client/agent/server application. The aglet is literally inserted into the path between the client and the server communicating with each other via messages. The message protocol is provided by the Aglet platform and is an RPC-like mechanism using Java sockets.

Thus, by using a parked DBMS-aglet, we avoid the reconnection cost (just like with the messenger approach)

3. The only difference is that the DBMS-aglet is database capable, in any other aspect they are identical.

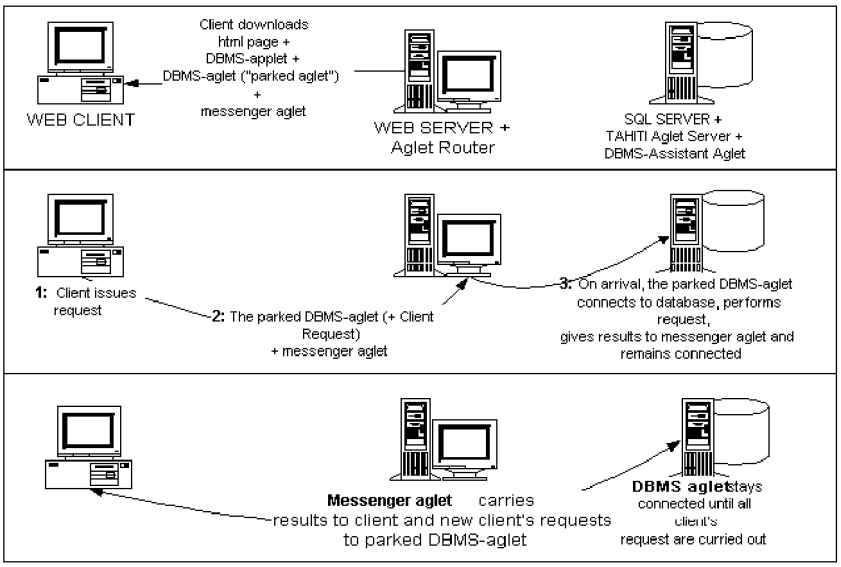

Fig. 3. DBMS-aglet framework using a messenger aglet.

and by using messages instead of the messenger aglet, we eliminate the time of negotiation and the amount of data transmitted between the client and the server. Performance results show that this approach by far outperforms the traditional applet approach as well as the other variations of the framework for direct data access.

3.4 Advantages of the DBMS-Aglet Framework

By using a DBMS mobile agent (namely the DBMS-aglet) to encapsulate all interactions between the client applet and the SQL server machine, the client applet becomes light and portable. This is achieved by:

- Avoiding the unnecessary downloading and initialization of JDBC drivers at the client's DBMS-applet.
- Passing the responsibility of loading the JDBC driver at the SQL server to the DBMS-aglet.
- Not using any JDBC API classes at the client's DBMS-applet.

The only responsibility of the client is to specify the URL address of the database server, the query to be performed, security certificates, and an itinerary. The rest is the responsibility of the DBMS-aglet. The effect on performance is quite significant, this delegation of responsibility results in performance gains of approximately a factor of ten (see performance evaluation in Section 5).

The DBMS-aglet is also independent of the various JDBC driver implementations. The DBMS mobile agent cannot (and is not supposed to) be aware of which JDBC driver to load when it arrives at an SQL server. Upon arrival at the SQL server's context, the DBMS-aglet is informed of all available JDBC drivers and corresponding data sources. The DBMS-aglet is then capable of attaching itself to one or more of these vendor data sources.

This is where the need for the local DBMS-assistant stationary aglet arises. The stationary aglet can be initialized and trained by the DBMS administrator to verify the certificates of incoming DBMS-aglets (thus, imposing the needed security) and to provide them with essential information about connecting to the local databases and carrying out their queries. The DBMS-assistant aglet can be made even more flexible. There might be cases, for example, where the DBMS-aglet encounters obstacles, such as an unreachable SQL server or an SQL server that fails to completely satisfy a user's query. If the DBMS-aglet is intelligent enough, it can negotiate with the local DBMS-assistant aglet to get alternative paths in order to accomplish its query.

The DBMS-aglet framework further benefits from the inherent advantages of the mobile technology itself:

- A DBMS-aglet can be fired (and forgotten) by a DBMS-applet initialized from a laptop or a palmtop computer during a short (and high-priced) Web session. The agent can roam around the unstructured network to perform the client's request and then wait until the communication link is again available to return home with the results.
- In cases of weak connectivity (i.e., a period of low bandwidth) between the client and the server, or when the client has limited storage capacity, a DBMS-aglet can minimize the transmitted information by performing both retrieval and filtering at the remote SQL server.
- A DBMS-aglet, having a certain load of work assigned to it, can split up the workload by cloning itself.
- A DBMS-aglet can dispatch to another host when its current host is powerless or has a very heavy workload.

In general, the DBMS-aglet framework allows aglets to be portable, light, independent, autonomous, flexible, and robust.

4 Using DBMS-Aglets to Query Multiple Database Systems

DBMS-Aglets can be launched to query multiple databases in parallel. The agents are launched to different hosts on the Web and they cooperate and communicate with each other

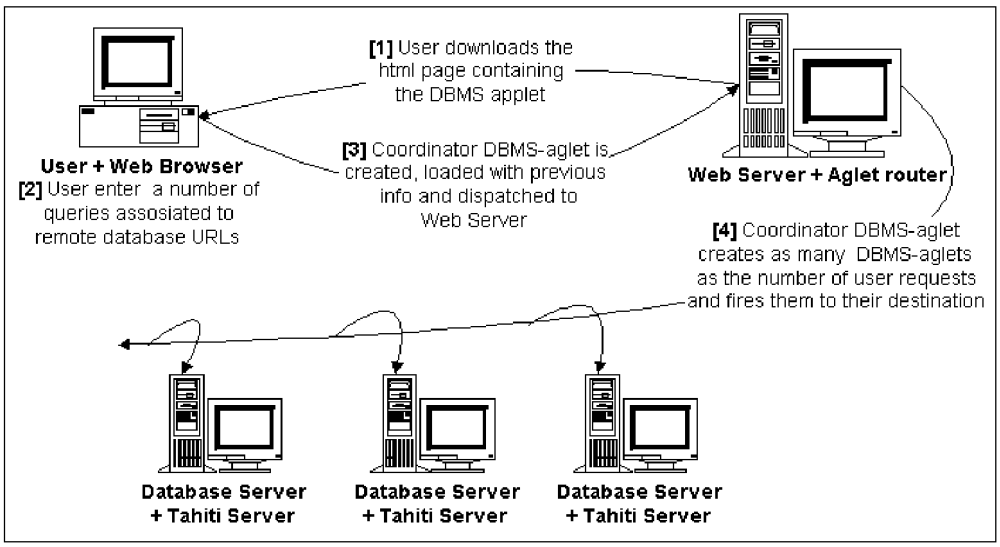

Fig. 4. The DBMS-agelt multidatabase framework.

to perform complicated tasks efficiently. There are various ways to use the DBMS-framework for querying multiple and possibly heterogeneous database systems. The straightforward way is for the DBMS-applet to create multiple DBMS-aglets, load each one of them with an SQL query and dispatch them to the various destinations. The DBMS-applet is responsible for combining the results provided by the various DBMS-aglets.

A more efficient approach is for the DBMS-applet to create an enhanced DBMS-aglet that is assigned the responsibility of creating the multiple DBMS-aglets, loading them with the queries, and dispatching them to their destination. The enhanced DBMS-aglet is also responsible for receiving and manipulating the intermediate results provided by the various DBMS-aglets. Only the final result is reported to the DBMS-applet. This approach is in line with our general objective of keeping the client light since the enhanced DBMS-aglet does not necessarily reside on the client. Processing can be done remotely with only the final result transmitted to the applet.

4.1 Demonstrating the DBMS-Aglet Multidatabase Framework

To realize the proposed framework, an enhanced DBMS-aglet, called Coordinator DBMS-aglet, is defined (see Figs. 5 and 6). The Coordinator DBMS-aglet creates and dispatches the DBMS-aglets, coordinates their execution, and composes their results. It is a direct descendant of the DBMS-aglet since it must have the capability of executing queries.

The basic functionality/responsibilities of a Coordinator DBMS-aglet are to:

- Maintain a list of aglet proxies, aglet Ids, and locations of the DBMS-aglets.
- Maintain a list of all the possible workstation's URLs that can host DBMS-aglets.
- Maintain a list of all the workstation's URLs that currently host DBMS-aglets.
- Create DBMS-aglets.
- Dispatch DBMS-aglets to workstations.
- Communicate with a specific DBMS-aglet, or broadcast a message to all DBMS-aglets that it has created.

The DBMS-applet creates a Coordinator DBMS-aglet that is dispatched to the fixed network most likely to the Web server. This aglet then dynamically creates and dispatches to several target workstations-hosts a variable number of DBMS-aglets to work in parallel (Fig. 4). The target workstations can be either computers in the fixed local network or computers connected through the Internet.

4.2 Refining the DBMS-Aglet Multidatabase Framework

In the cases where we anticipate the processing of multiple sets of queries, as for instance, in the case of virtual enterprises [29], the Coordinator DBMS-aglet can be "extended" (see Fig. 5) to create and submit "parked" DBMS-aglets instead of DBMS-aglets (and, as it is shown in Fig. 5, create the "Coordinator-Aglet (Blue)"). Thus, the first set of queries creates a network of parked mobile agents. The Coordinator DBMS-aglet can use the infrastructure of agents already set-up to process any subsequent set of queries. If a new site is to be accessed for one of these queries, the coordinator aglet might choose, based on some heuristics, not to create a new aglet but to instruct a nearby one to move to the new site and execute the query. The SQL query is sent to this aglet along with the "move" instruction. For reference purposes, we call this coordinator aglet the "intelligent" coordinator aglet.

Parallel execution of queries at multiple sites often requires combining or comparing the various results in specific ways. Certain applications might require the coordinator to integrate or simply compare (e.g., for the maximum or minimum value) the results returned by the various agents before transmitting them to the client. This essentially introduces another computational task for the Coordinator DBMS-aglet: combining or comparing the various results. Aglets can employ multithreading to receive and manipulate the various results in parallel. Multithreading capabilities and the extra tasks can be easily added by extending the basic Coordinator DBMS-aglet (see Fig. 5).

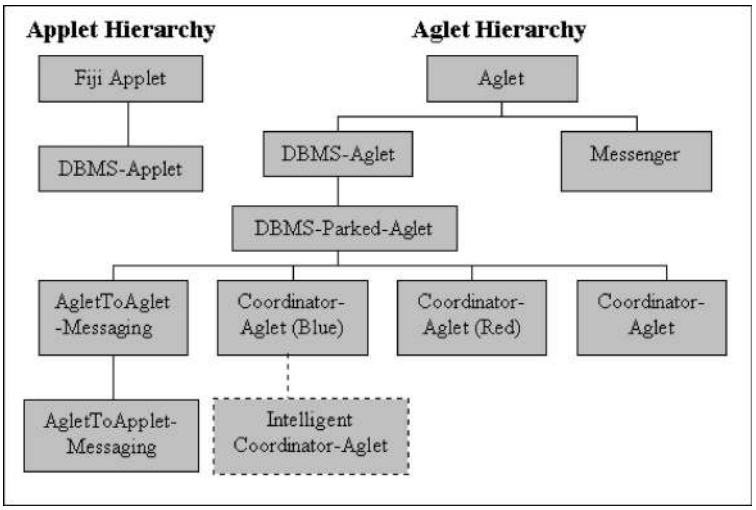

Fig. 5. The DBMS-aglet framework's java object hierarchy.

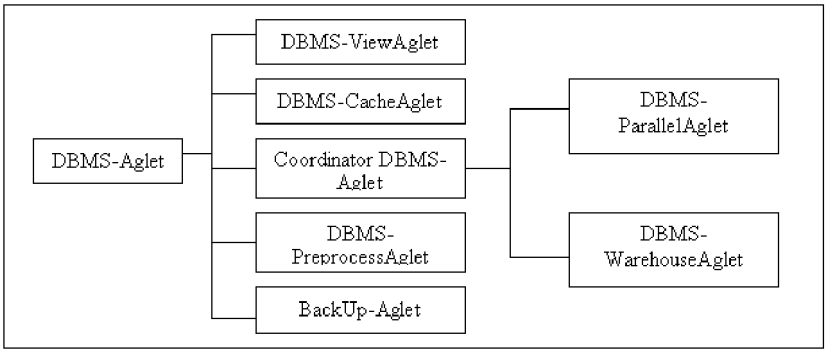

Fig. 6. This is an example of the Visionary library.

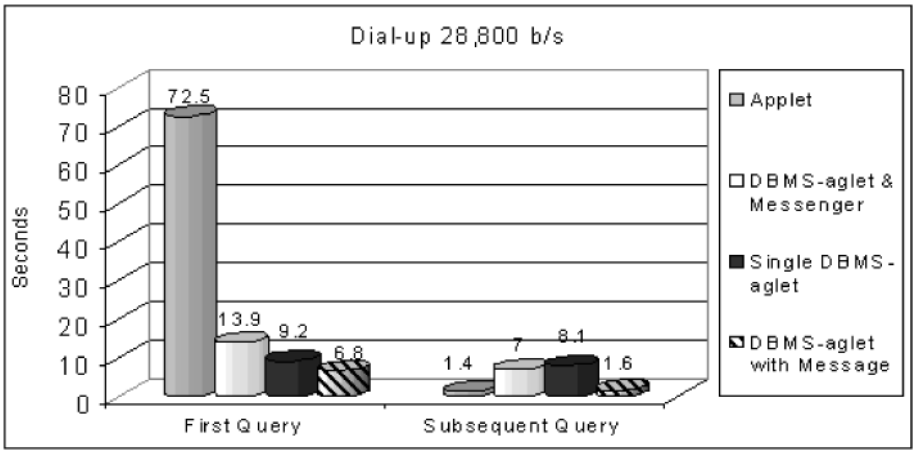

Fig. 7. Mean times for 28,800 b/s client connectivity.

4.3 Advantages of the DBMS-Aglet Multidatabase Framework

When compared with the traditional applet approach to accessing multiple databases, the DBMS-aglet multidatabase framework offers significant advantages. For the traditional applet approach to work, the applet must a priori know the JDBC driver for each one of the remote databases to be accessed. Thus, the applet cannot dynamically increase or modify the set of databases to be queried. On the contrary, such information is not required when the DBMS-aglet framework is employed. The number of DBMS-aglets in the DBMS-aglet framework can change dynamically. After initiating the application and the coordinator DBMS-aglet, the coordinator can create, at run time, as many DBMS-aglets as the application requires. After that, additional DBMS-aglets can be created as needed. The databases that will host the DBMS-aglets can also be decided upon at run time, after considering the network traffic or the workload of the available workstations.

Another shortcoming of the applet model that the DBMS-aglet framework bypasses is that in the applet

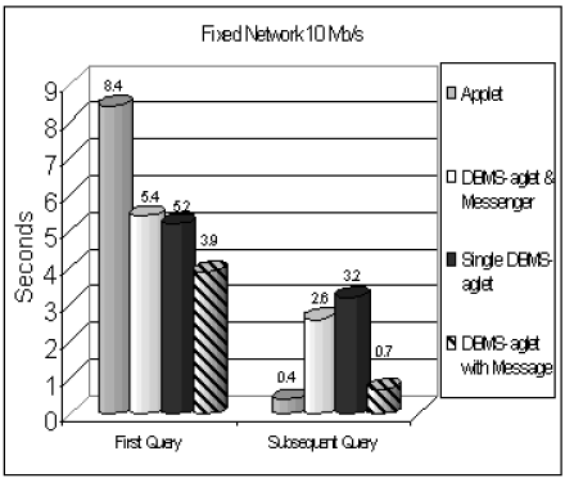

Fig. 8. Mean times for 10 Mb/s client connectivity.

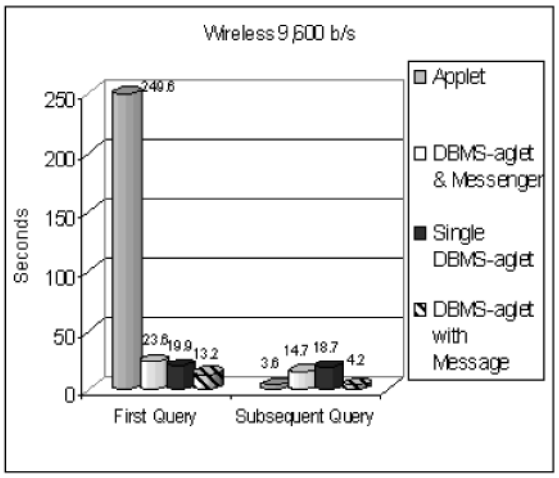

Fig. 9. Mean times for 9,600 b/s client connectivty.

approach, the client must download a large number of JDBC drivers; one for each remote database to be accessed. This is very costly, severely degrading the applet's performance. In the aglet approach, the client is relieved from this overhead since downloading any JDBC drivers is the responsibility of the various DBMS-aglets.

5 A Specialized-Agents Library

Fig. 5 shows the Java hierarchy of the aglets composing the various DBMS-Aglet frameworks. The aglets of our framework have been created by expanding the capabilities of the basic DBMS-aglet. For example, to utilize the "message" DBMS-Aglet Multidatabase Framework, one has to include in the DBMS-applet only the "Coordinator-Aglet (Blue)." This aglet will use the AgletToAglet-Messaging aglet which is (as shown in the hierarchy) a DBMS-Parked-Aglet (i.e., "parked" DBMS-aglet) that can communicate via messages with aglets. The AppletToAglet aglet can communicate with DBMS-Applets. The example presented in Section 4.1 utilizes the "Coordinator-Aglet" (see Fig. 5) which creates, dispatches, and receives DBMS-aglets. The "Coordinator-Aglet (Red)" utilizes the DBMS-Parked-Aglet and the messenger aglet. Similarly, the "intelligent" Coordinator DBMS-aglet can be easily created by extending, for example, the "Coordinator-Aglet (Blue)."

A Java mobile execution environment extends the scope boundaries of object-oriented programming over the network. An aglet can be viewed as the basic abstract object that provides basic capabilities such as mobility, communication with other aglets, and self-cloning. The DBMS-Aglet Framework enhances these capabilities with the addition of efficient database connectivity. Then, the DBMS-Applet and the DBMS-Aglet can be seen as abstract classes. To further enhance these new aglets with other capabilities is now very simple; one can extend them (via inheritance) to include the desired functionality. For example, we can extend the DBMS-Aglet to create a DBMS-ViewAglet capable of materializing a requested or personalized [30] view, a DBMS-CacheAglet, a DBMS-PreprocessAglet, or a DBMS-ParallelAglet with parallel processing capabilities. Fig. 6 shows a possible aglets' hierarchy. Our goal is to extend the DBMS-Aglet Framework to create a complete library of database aglets from which a user can pick and instantiate the aglet required by a specific application.

6 Performance Evaluation

The performance evaluation compares the total time required by a Web client to access and query a remote database between the traditional applet-based and the three proposed DBMS-aglet approaches. In more detail, we are interested in the time required, for each approach, to query the remote database for the first time and for any subsequent requests. We consider both short (composed of three queries) and long (composed of six queries) transactions between the client and the remote database. Short or long transactions are the type of transactions generally anticipated by Web users.

For each approach, we performed the evaluation having the client accessing the Web server via:

- A 28,000 b/s dial-up connection to an Internet Service Provider (ISP).
- A 9,600 b/s wireless dial-up connection to an ISP.
- A 10 Mb/s Ethernet connection (fixed network).

For each approach and client connectivity case, we performed the tests numerous times and from different remote clients. Specifically, each set of experiments consisted of more than 100 queries randomly distributed between the seven hourly intervals composing the time span between 9 a.m. and 5 p.m.. Each tested approach provides two data sets of results (observations); one for the first query and one for the subsequent queries.

6.1 The DBMS-Aglets

We used a Borland provided JDBC type 3 driver[4] and performed the tests over Borland's Paradox database. For the DBMS-aglet approach since the connection is performed locally, we used the local version of the JDBC driver.[5] The configuration used includes a Web server, a remote

4. The driver is called borland.jbdc.Broker.RemoteDriver. This driver speaks to a Gateway program installed at the Web server, called Data Gateway (also provided by Borland), which translates its request into paradox SQL commands.

5. This local driver is called, jdbc.Bridge.LocalDriver, it translate the JDBC API into paradox SQL commands.

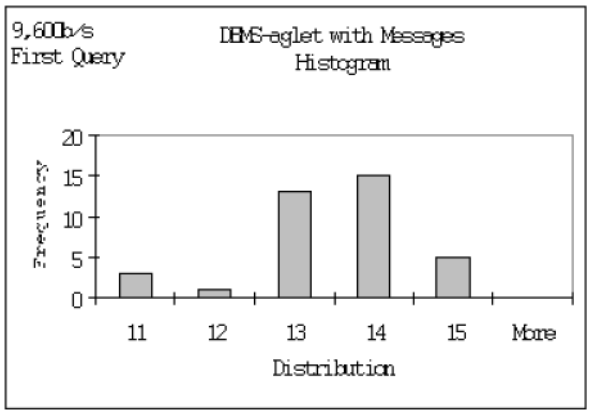

Fig. 10. Distribution in the DBMS-aglet with Messages approach, first query.

Fig. 11. Distribution in Traditional applet approach, first query.

relational database (namely Borland's Paradox), and a Windows 95 client. The degree of distribution employed is a bit limited, confined only within the boundaries of a small geographic area. We expect the results to be more favorable for mobile agents in a wide area networking setting. For the fixed network case, however, we performed the test within the larger University network.

For the applet-based approach, we 1) installed at the Web server a type 3 JDBC driver, 2) created an applet responsible for accessing and querying the remote database, called db-applet, and 3) installed at the Web server an html page containing the db-applet. The time measured for the first query includes the time required:

- by the JDBC driver to be downloaded and initiated at the db-applet,
- by the db-applet to connect to the database,
- by the db-applet to query the database.

We then measured the time required by the db-applet to issue a subsequent request.

For the DBMS-aglet approach, we performed three sets of tests, one for the single aglet method, one for the two aglets (i.e., the parked and the messenger aglet), and one for the message based method. The time measured for the first query includes the time required:

- by the DBMS-aglet (+ messenger for the second method) to be dispatched from the DBMS-applet to the database server,
- by the DBMS-aglet to locally connect and query the database,
- by the DBMS-aglet (or messenger for the second method or message for the third approach) to fetch the result back to the DBMS-applet.

The total time measured for any subsequent query includes the time required:

- by the DBMS-aglet (or messenger for the second method or message for the third approach) to be dispatched from the DBMS-applet to the database server carrying the client's next request,
- by the DBMS-aglet to locally (re)connect and query the database (only for the single aglet approach),
- by the parked DBMS-aglet to receive from the messenger (or message) the query and execute it (only for the second and third methods),
- by the DBMS-aglet (or messenger or message) to fetch the result back to the DBMS-applet.

6.1.1 Performance Statistical Analysis

The first step of the data analysis was to perform a descriptive statistical analysis. This analysis gave information about the behavior of the data sets involved in the statistical analysis, including a description of the mean, the median, the standard deviation, the kurtosis, and the skewness coefficients. In particular, a comparison of the means and the standard deviations has been performed indicating the most efficient approach, which seems to be the "Parked DBMS-aglet with Messages" for the first queries and the "Traditional Applet" method for subsequent queries. However, the marginal difference between the performance of the two methods for subsequent queries is compensated by the significant difference in the performance for first queries, suggesting that the "Parked DBMS-aglet with Messages" could be considered as the most efficient approach for all cases of client connectivity. Figs. 7, 8 and 9 show the mean time required for all approaches.

What was observed by the descriptive statistical analysis was also verified by the construction of histograms, which lead us to important conclusions regarding the spread of each approach. For example, as shown in Figs. 10 and 11 (histograms) with a client connectivity of 9,600 b/s for the first query, the spread of time required for the traditional DBMS applet is quite wide having a variance value of 612.34, whereas for the "DBMS-aglet with Messages" approach, the spread is quite limited with a variance of 1.0359. This wide spread in the Applet approach proves that the usual time required for a transaction to be committed is far away from being stable and lies between a wide range of time intervals, where as the "DBMS-aglet with Messages" approach guarantees that the time required for a transaction to be executed lies between much shorter time barriers, and is considered to be much more stable. This behavior is possibly attributed to the fact that the applet approach downloads the various JDBC classes on a need to have basis, thus continuously subjecting itself to the

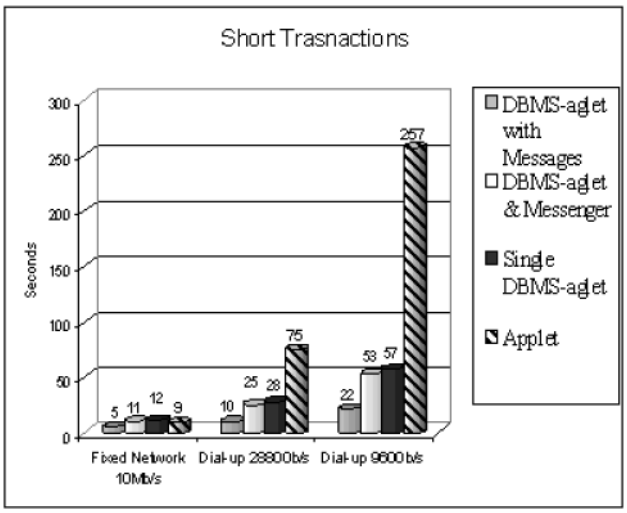

Fig. 12. Short transactions graph.

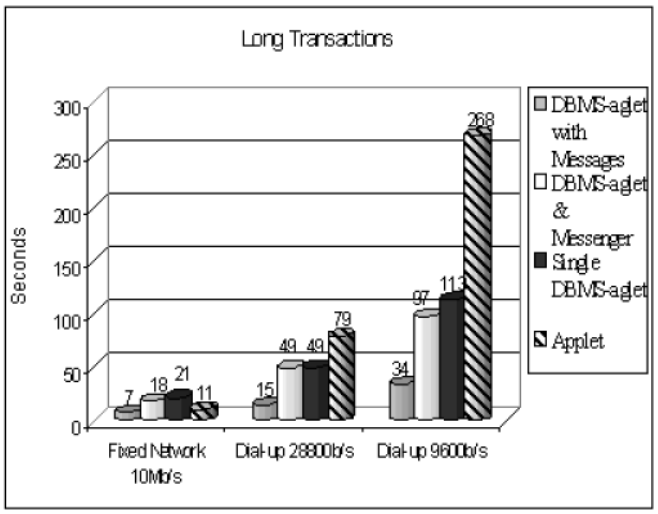

Fig. 13. Long transactions graph.

communication link's random behavior. Having the agent residing on the remote and powerful database server, we avoid this disturbing fluctuation since we minimize the client's communication.

The lack of outliers in the data sets was suggested by a comparison of the mean and median values of each data set, indicating that the difference between the two values is insignificant. The latter result allowed us to consider the mean of the samples (tests) taken as a reliable measure for continuing the statistical analysis. On the other hand, a correlation analysis showed the lack of any linear dependency between any two approaches.

The reliability of the mean as a measure of tendency for each sample and the lack of any linear dependencies among the approaches, as well as the number of observations (measurements) (which come up to 30 or more), allow us to use the Z-test [23] in order to compare the means of the populations of interest, using the information of the corresponding samples to a certain degree of certainty. In our case, the population of interest is the set of all the real times in seconds required by a client to query the remote database using one approach, either for first or for subsequent query, with a certain client connectivity.

The Z test proved that the time required for the "DBMS-aglet with Messages" approach for first queries is always less than the time required for any other approach (see full version [24]). A similar analysis has been made for all approaches concerning subsequent queries proving that the best approach for all cases of the client connectivity is the "Traditional Applet." The difference, however, from the "DBMS-aglet with Messages" approach is insignificant. Taking the results from the Z test, we safely produced the time required for short and long transaction for each approach under a given Web client connectivity case. A short transaction's cost is the sum of 1) the time required for the first query and 2) the time required for two subsequent requests. For a long transaction, item 2 is the time required for five subsequent requests. Figs. 12 and 13 illustrate the time required for short and long transactions, respectively.

As shown in both graphs, the "DBMS-aglet with Messages" approach requires considerably less time than any other approach, and the applet approach significantly more than any other does, except in the fixed network case. In fact, in a wireless and dial-up environments, this variation of the framework provides a performance improvement of approximately a factor of ten. For the fixed network, the gains are about 40 percent and 30 percent, respectively.

6.2 The Multidatabase DBMS-Aglets

6.2.1 *Prototype Implementation: Accessing Multiple Databases*

The goal of this application is to provide the user with the capability of querying a number of distributed heterogeneous databases and joining the result tables of each query to produce the final result. The application has been tested for two different configurations; one consisting of a set of three and one consisting of a set of six remotely located databases.

The experiment began with the user initiating (by downloading the DBMS-applet), within the context of the local Tahiti server, the Coordinator DBMS-aglet. Through the applet's graphical interface, the user issued a set of SQL subqueries and the IP addresses of various hosts where the databases are located. When the user enters the execute command, the Coordinator DBMS-aglet moved to the fixed network (e.g., on the Web server), created three/six DBMS-aglets, and gave them vital information including:

- Credentials, passwords, etc.
- The IP address of the remote host to visit.
- The query to execute with the database server on each host.
- The IP address to return to the Coordinator DBMS-aglet.

Each DBMS-aglet is then dispatched to the remote hosts carrying along the above information. Arriving at the hosts, each DBMS-aglet presented its credentials, connected to the local SQL database server, and performed its query. As soon as the results were available, each DBMS-aglet dispatched home (their home is the Web server) with the results (a database table). Arriving back home, each DBMS-aglet

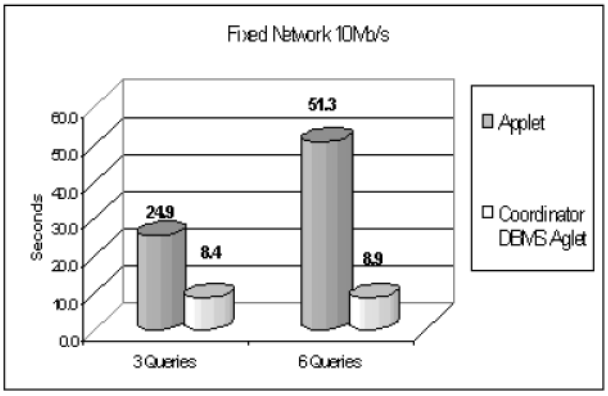

Fig. 14. Multidatabase DBMS-aglets vs. Traditional multidatabse applets.

delivered the results to the Coordinator DBMS-aglet. The Coordinator DBMS-aglet, using a Java-thread based approach, joined the various results and delivered the final outcome to the client.

6.2.2 Performance Comparison

We have performed a statistical analysis similar to the statistical analysis we performed for the single DBMS-aglet approach. The Multidatabase DBMS-aglet approach, as shown in Fig. 14, by far outperforms the traditional applet approach. Fig. 14, due to space consideration, only presents the fix network comparison since it is the only environment the applet approach could fare positively. In the other two environments, the aglet framework, as it could be easily projected from the single database approach (see Section 6.1), completely outstrips the traditional applet approach.

The multidatabase aglet approach performs extremely well since it does not have to deal with the downloading of the various JDBC drivers and its interaction with the client is minimized to transmitting the final result. The performance does not significantly change with the number of queries since the DBMS aglets are sent to the various destinations to work in parallel. For the reverse reasons, the applet approach shows a linear degradation in performance.

7 Implementation Issues

7.1 The Role of the Implementation Platform

In this work, we have used IBM's Aglets Workbench [7]. Aglets have mainly concentrated on functionality and not, thus far, on performance [28]. To give an example, in the Aglets workbench, whenever an agent is transferred to a new destination, all the reachable objects are transported along with it. This is done even in the cases where the agent is sent to the same destination multiple times. Other Java-based mobile agents platforms include ObjectSpace's Voyager [22], Mitsubishi's Concordia [27], IKV++ Grasshopper [25], and General Magic's Odyssey [26]. While they are all based on Java, each one of them adds its own special implementation features that can significantly affect performance.

In this section, we report on our experiments in implementing our framework using a different agent

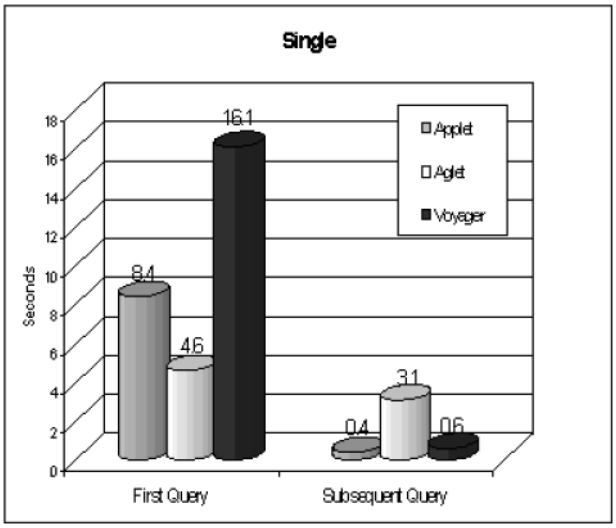

Fig. 15. Voyager vs. Aglet for the "Single" approach.

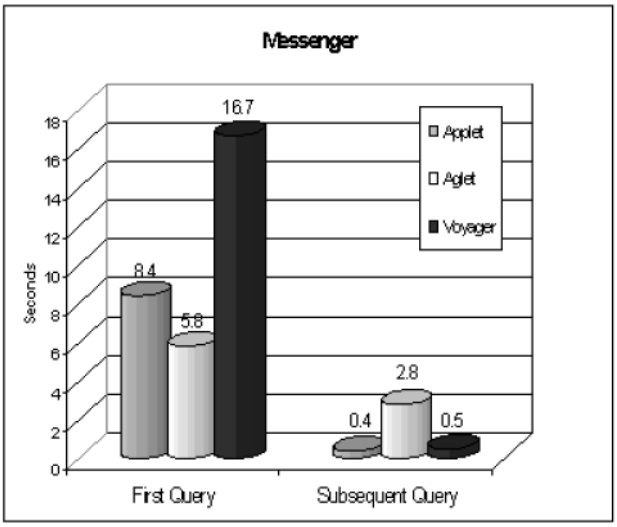

Fig. 16. Voyager vs. Aglet for the "Messenger" approach.

platform, namely Voyager. The results show that the new implementation of our framework outperforms the applet approach, even in the case of the fixed network.

We performed the same tests under the same network/system configuration, as well as the needed statistical analysis. Worth noting is that the porting of our framework in the Voyager platform was performed smoothly and in a timely fashion. Figs. 15, 16, and 17 compare the Voyager implementation with the Applet and Aglet approach for all the proposed frameworks.

In all approaches, for the subsequent query, the Voyager implementation by far outperforms the Aglet implementation. For the "single" (see Fig. 15) and the "messenger" (see Fig. 16) approaches the performance is very close to that of the Applet approach. The "message" (see Fig. 17) approach, however, outperforms the Applet approach by a factor of thirteen. This improvement is attributed to Voyager's agent transportation procedure. Voyager is based on RMI, while Aglets implement their own agent transport protocol (ATP). Aglets, in each and every agent transport, transmit all the needed class objects as well, while Voyager transports

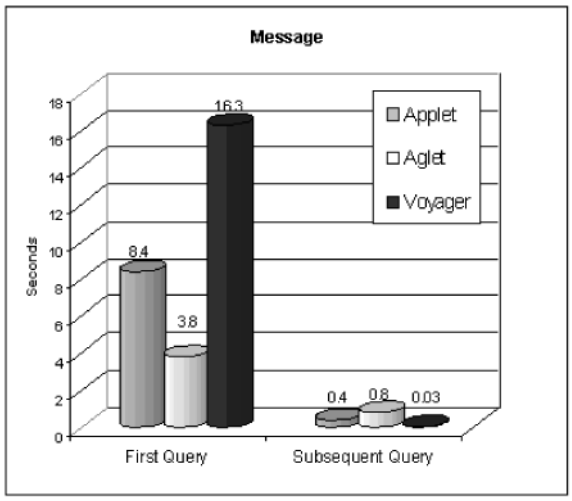

Fig. 17. Voyager vs. Aglet for the "Message" approach.

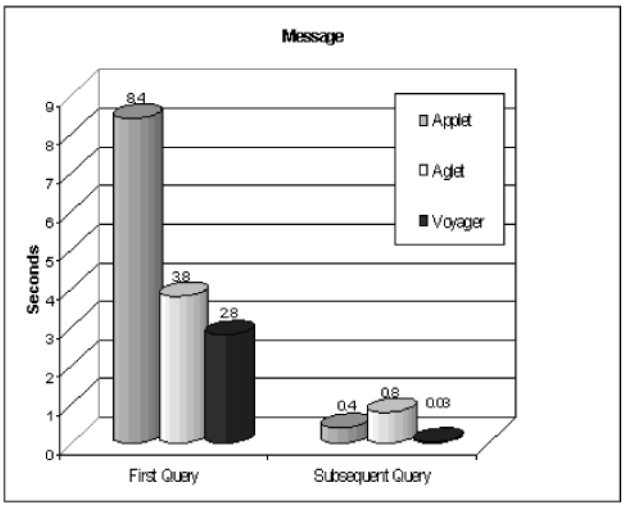

Fig. 18. Voyager special implementation of the "Message" approach.

(following the RMI philosophy) the needed object classes only once and on a need-to-use base. Any subsequent request is satisfied by objects now cached at the destination.

The performance of the first query, however, is disappointing. In every approach, the Voyager implementation performs the worst; see Figs. 15, 16, and 17. This "bad" performance is the trade off for higher flexibility. This difference in performance is attributed to the fact that during the execution of the first query, the Voyager implementation requires the downloading (to the client) of all the needed classes for the dynamic creation of the agent execution environment. Aglets do not need to do that since the required classes are preloaded to the client (the so called aglet plug in).

Mobile agents, however, provide unique flexibility allowing alternate ways of approaching the same problem. The result of this flexibility is shown in Fig. 18. In the chart, the Voyager implementation is shown to outperform both the Aglet and Applet approaches for the first, as well as the subsequent, query. This was achieved by creating the "parked" agent on the Web server and not on the client, thus avoiding the need to download the agent execution environment on the client. The agent execution environment is created locally on the Web server. On the client, we only send the proxy of the "parked" agent to enable the communication of the DBMS-applet with the "parked" agent. The client gets the user request, composes the SQL query and, alone with the other needed information, submits it via a message to the proxy, and, hence, to the "parked" agent. The "parked" agent receives the information (i.e., the Database's URL, the SQL query, etc.) and moves to park and execute it at the remote destination.

7.2 Scalability

Our experimental results show that (as expected) which mobile agent platform is used to implement our framework affects its performance. For instance, in the particular case of Fig. 18, the fact that the Voyager mobile agent platform provides the ability to remotely create an agent, resulted in an efficient implementation that outperformed all other approaches. The fact that this or other features are or are not supported by the Aglets framework is not addressed further here since 1) such features can be easily added to the Aglets framework[6] and 2) our focus in this paper is to show how mobile agents can be used to efficiently support web access to remote databases and not a thorough evaluation of the existing mobile agent platforms.

Such results can be found in [31], [32], where we have studied and compared the various Java-based mobile agent platforms (i.e., Aglets, Voyager, Concordia, and Grasshopper) based on a number of mobile-agent specific benchmarks that we have developed. Agent transportation, multithreading, CPU timeslicing, message queuing, and communication protocols are some of the issues affecting the performance, efficiency, and scalability of an agent platform.

Another issue that heavily depends on the agent execution environment and its implementation is scalability, both in terms of the number of concurrent clients and the size of the results. In general, in terms of the number of mobile agents that they can efficiently support, Aglets scored at the lower end while Voyager at the higher end of the scale [31], [33]. Initial scalability tests of the DBMS-Aglet frameworks agreed with those results, indicating that the Aglets implementation of the framework does not scale very well. Results, though, that tested scalability based on the size of the result set, have shown that the Aglets Framework scales better than the traditional applet approach [33].

7.3 Dynamic Configuration

An important advantage of the various DBMS-Aglet frameworks is that they can be configured dynamically. Currently, the approach chosen by various database vendors (e.g., Oracle [34]) to employ the client/agent/server (c/a/s) model, is to have appropriate components configured and set up a priori for each different application. This results in a static configuration. Accessing a new site becomes, if not impossible quite cumbersome, prohibiting

6. In fact, with the addition of an extra class library, the "Aglet.com," the current version of Aglets can provide a somewhat inflexible version of this feature.

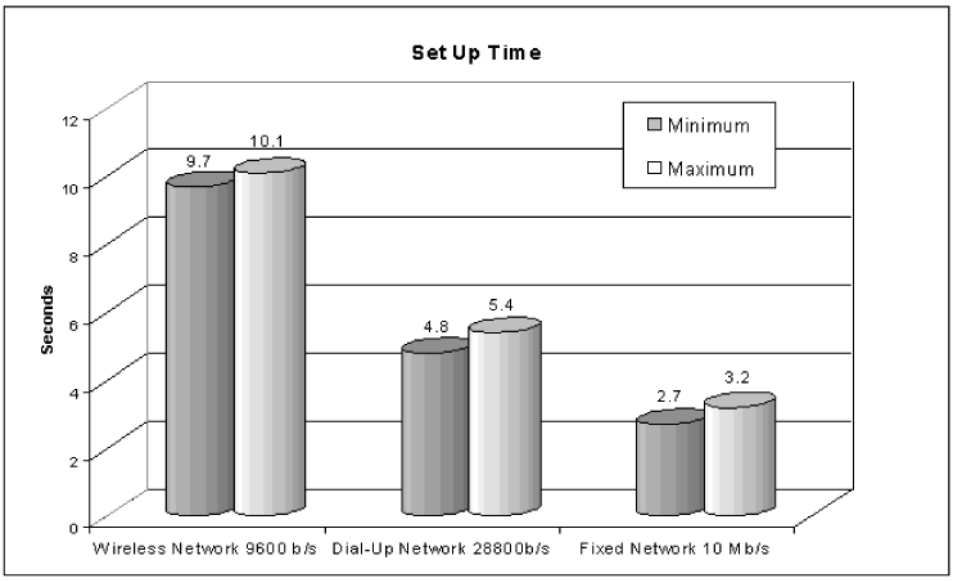

Fig. 19. Time needed to set up the c/a/s model(s) for the wireless, dial-up, and the fixed network. The set up times are the same for both the "message" and "messenger" approach since only a single agent is sent to the fixed network.

TABLE 2
Fixed Networks 10MBits

Fixed Network 10MBits

	Applet - First Query	*Applet - Subsequent Query*	*Messenger - First Query*	*Messenger - Subsequent Query*	*No Messenger - First Query*	*No Messenger - Subsequent Query*	*Message - First Query*	*Message - Subsequent Query*
Mean	8.4	0.4	5.4	2.6	5.2	3.2	3.9	0.7
Standard Error	0.1596196	0.0488584	0.1396806	0.1110031	0.0727376	0.1189574	0.0240615	0.0121614
Median	8	0.4	5.5	2.4	5.05	2.9	3.9	0.7
Mode	8	0.3	5.5	1.5	4.8	2.4	3.9	0.7
Standard Deviation	1.1944841	0.3752887	1.0072512	0.8669619	0.5443184	0.9290867	0.1178767	0.0719477
Sample Variance	1.4267922	0.1408416	1.0145551	0.751623	0.2962825	0.8632022	0.0138949	0.0051765
Kurtosis	7.9228303	38.838444	1.0348965	0.3038037	0.2946658	-0.8796017	-0.2094634	2.1892609
Skewness	2.8153771	5.6830863	0.8652408	0.9510618	0.8926834	0.4863531	-0.7763496	-0.3216294
Range	6	2.9	4.5	3.6	2.4	3.6	0.4	0.4
Minimum	7	0.1	4	1.5	4.4	1.9	3.6	0.5
Maximum	13	3	8.5	5.1	6.8	5.5	4	0.9
Sum	471	25.5	282.2	158.2	293.5	198.1	93.1	25.2
Count	56	59	52	61	56	61	30	35
Largest(1)	13	3	8.5	5.1	6.8	5.5	4	0.9
Smallest(1)	7	0.1	4	1.5	4.4	1.9	3.6	0.5
Confidence Level(95.0%)	0.3198848	0.0978007	0.2804203	0.2220392	0.1457694	0.2379501	0.0497749	0.0247149

global utilization of network resources. To access a server at a network site, the site must be appropriately configured in advance to include necessary software modules. Combining the c/a/s model with the capabilities provided by mobile agents permits dynamic configuration of the model.

The "messenger" approach can be viewed as the adaptation of the c/a/s model, where the communication with the server is done via agents, one could call it "c/a/s-MA." On the other hand, the "message" approach can be viewed as the direct materialization of the c/a/s model. We are mostly interested in the "messenger" and "message" approaches since these are the direct adaptations of the c/a/s model. These approaches can allow for dynamically configuring applications to follow each of the c/a/s models. Such dynamic generation of the c/a/s software model offers flexibility, adaptability, and ease of use. Note that we can dynamically send and install the DBMS-assistant stationary aglet where needed as well. Furthermore, the mobile agent implementation/materialization of this model(s) further enhances its applicability to mobile wireless computing, making applications more light-weight, tolerant to intermittent connectivity, and adaptable to dynamically changing environments.

TABLE 3
Dial up 28,800

Dial up 28800

	Applet - First Query	*Applet - Subsequent Query*	*Messenger - First Query*	*Messenger - Subsequent Query*	*No Messenger - First Query*	*No Messenger - Subsequent Query*	*Message - First Query*	*Message - Subsequent Query*
Mean	72.5	1.4	13.9	7	9.2	8.1	6.8	1.6
Standard Error	0.7835293	0.1541088	0.1111488	0.1208596	0.1414027	0.1327706	0.0129891	0.0156125
Median	71	0.8	14	7.1	9	8.1	6.8	1.5
Mode	70	3	14	7.6	9.5	8	6.8	1.6
Standard Deviation	5.810809	1.2036284	0.7780415	0.8798709	0.979667	0.9846533	0.09633	0.1219379
Sample Variance	33.765502	1.4487213	0.6053486	0.7741727	0.9597473	0.9695421	0.0092795	0.0148689
Kurtosis	3.7560885	6.6316481	0.01216	-0.3019246	3.5296042	-0.7037168	0.9681475	0.9378452
Skewness	2.0007924	2.2322735	0.530405	-0.6129948	1.155589	-0.2170342	-0.26327	0.8819108
Range	26	6.5	3.2	3.5	5.5	3.7	0.5	0.5
Minimum	64	0.5	12.8	5.1	7.5	6.3	6.5	1.4
Maximum	90	7	16	8.6	13	10	7	1.9
Sum	3990.1	83.1	680.7	371.4	441.3	443.1	374.7	94.7
Count	55	61	49	53	48	55	55	61
Largest(1)	90	7	16	8.6	13	10	7	1.9
Smallest(1)	64	0.5	12.8	5.1	7.5	6.3	6.5	1.4
Confidence Level(95.0%)	1.5708831	0.3082634	0.2234795	0.2425223	0.2844654	0.2661893	0.0260417	0.0312297

TABLE 4
Wireless 9,600

Wireless 9600

	Applet - First Query	*Applet - Subsequent Query*	*Messenger - First Query*	*Messenger - Subsequent Query*	*No Messenger - First Query*	*No Messenger - Subsequent Query*	*Message - First Query*	*Message - Subsequent Query*
Mean	249.6	3.6	23.6	14.7	19.9	18.7	13.2	4.3
Standard Error	4.9491009	0.5594901	0.2231406	0.1385043	0.2026549	0.0920721	0.1673252	0.0781048
Median	250	1.92	24	14.9	20	18.6	13.6	4.25
Mode	230	1.7	24	14	19	19	14	3.9
Standard Deviation	24.745505	3.4940143	1.1377981	0.807612	1.128335	0.512636	1.0177995	0.4814703
Sample Variance	612.34	12.208136	1.2945846	0.6522371	1.2731398	0.2627957	1.0359159	0.2318137
Kurtosis	1.674639	0.5275049	-0.6964242	0.6315163	-0.6270616	-0.1492015	-0.1686204	-1.2058077
Skewness	1.1455276	1.5454641	0.0975586	0.5281871	0.4267024	0.3381334	-0.5866162	0.3452634
Range	101	10.5	4	4	4	2	4	1.6
Minimum	219	1.5	22	13	18	18	11	3.7
Maximum	320	12	26	17	22	20	15	5.3
Sum	6239	142.14	614.8	501.3	616.2	578.8	489.4	164.1
Count	30	39	30	34	31	31	37	38
Largest(1)	320	12	26	17	22	20	15	5.3
Smallest(1)	219	1.5	22	13	18	18	11	3.7
Confidence Level(95.0%)	10.21444	1.1326285	0.4595663	0.2817894	0.4138762	0.1880362	0.3393508	0.1582552

Fig. 19 summarizes the time needed to set up each of the models. For the wireless and dial-up cases, this time is 9.7 to 10.1 seconds and 4.8 to 5.4 seconds, respectively. In the wireline case, to set up the various models takes between 2.7 and 3.2 seconds. Note that the times recorded in Section 5 for the first query includes the setup time as well. We considered this appropriate since we wanted to charge for the dynamic installation of the models.

7.4 Other Approaches

Another popular class of approaches, called Server Side Include (SSI) [35], [26], suggest the use of scripting languages inside html code. Such a scripting language defines the parameters for a database transaction between

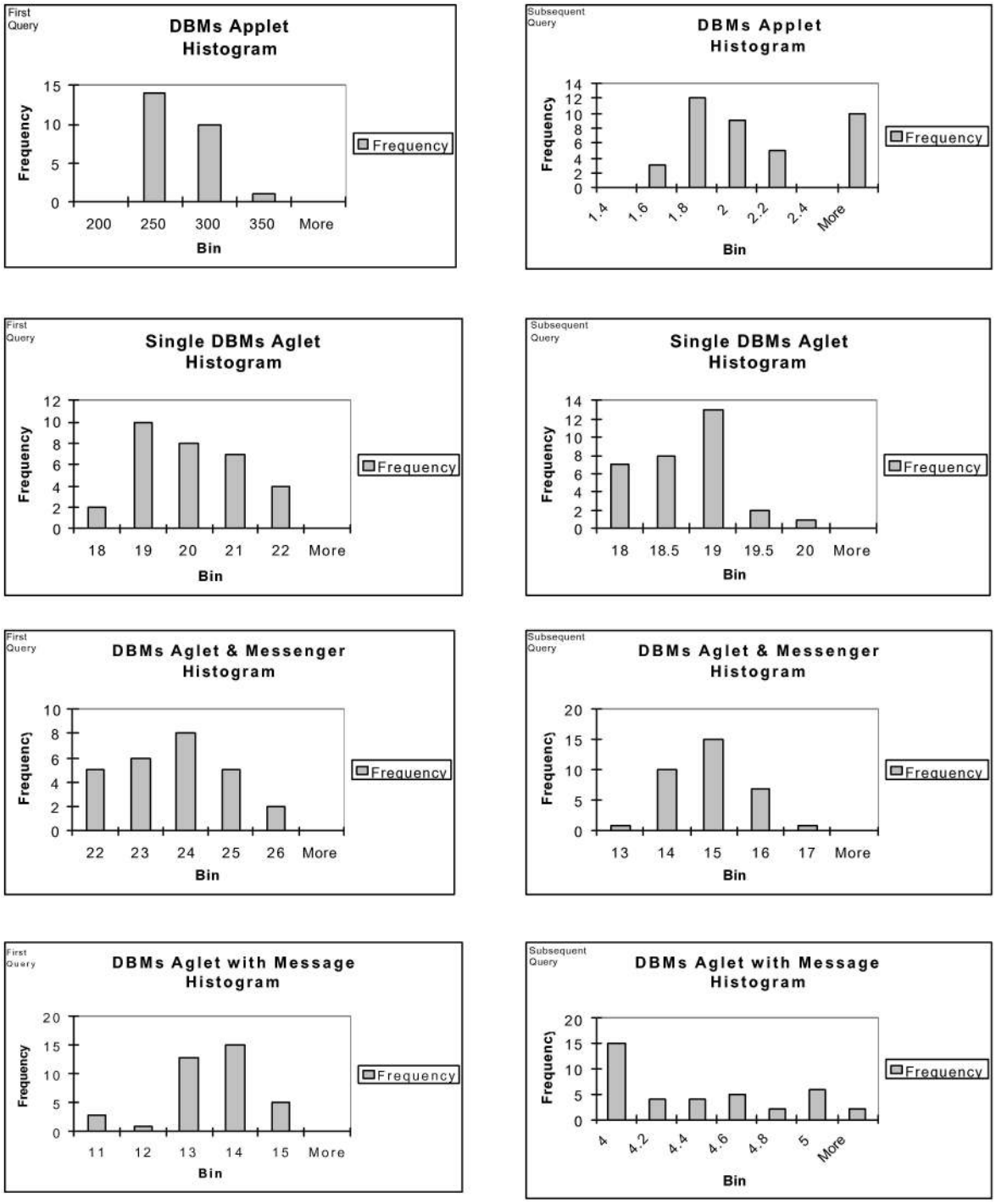

Fig. 20. Histograms for Wireless 9600 b/s client connectivity. A similar analysis for the Fixed Network 10 Mb/s and the Dial up 28,800 can be found in [24] or www.ada.cs.ucy.ac.cy/~cssamara/dbms-agents.

the database server and the Web browser. In this case, the Web server program must be modified to interpret the scripting language. The idea is simple and easy to implement, but it suffers from the fact that each database transaction requires the opening of a new connection with the database Server (like CGI). In addition, it suffers from compatibility problems since the use of a dedicated web Server program is required. The model following by these approaches is the client/agent/server model (or c/a/s like model) and, thus, they are additionally suffer from all the problems presented in the previous subsection.

8 Conclusions

Web technology advanced distributed systems by providing the underlying fabric for communication among participating processes located at remote computers connected through the Internet. In this paper, we have introduced a new approach for accessing remote databases from the Web using Java mobile agents. The proposed "DBMS-Aglet Framework" supports light-weight, portable, and autonomous clients as well as operation on slow or

Wireless 9600

First Queries Comparison

Applet vs Messenger
z-Test: Two Sample for Means

	Variable 1	*Variable 2*
Mean	249.56	23.64615
Known Variance	612.34	1.29
Observations	25	26
Hypothesized Mean Difference	1	
z	45.39944	
P(Z<=z) one-tail	0	
z Critical one-tail	1.644853	
P(Z<=z) two-tail	0	
z Critical two-tail	1.959961	

Messenger vs No Messenger
z-Test: Two Sample for Means

	Variable 1	*Variable 2*
Mean	23.64615	19.87742
Known Variance	1.29	1.27
Observations	26	31
Hypothesized Mean Difference	1	
z	9.199361	
P(Z<=z) one-tail	0	
z Critical one-tail	1.644853	
P(Z<=z) two-tail	0	
z Critical two-tail	1.959961	

No Messenger vs Message
z-Test: Two Sample for Means

	Variable 1	*Variable 2*
Mean	19.87742	13.22703
Known Variance	1.27	1.04
Observations	31	37
Hypothesized Mean Difference	1	
z	21.49886	
P(Z<=z) one-tail	0	
z Critical one-tail	1.644853	
P(Z<=z) two-tail	0	
z Critical two-tail	1.959961	

Subsequent Queries Comparison

Applet vs Messenger
z-Test: Two Sample for Means

	Variable 1	*Variable 2*
Mean	3.644615	14.74412
Known Variance	12.21	0.65
Observations	39	34
Hypothesized Mean Difference	1	
z	-20.9928	
P(Z<=z) one-tail	0	
z Critical one-tail	1.644853	
P(Z<=z) two-tail	0	
z Critical two-tail	1.959961	

Applet vs No Messenger
z-Test: Two Sample for Means

	Variable 1	*Variable 2*
Mean	3.644615	18.67097
Known Variance	12.21	0.26
Observations	39	31
Hypothesized Mean Difference	1	
z	-28.2663	
P(Z<=z) one-tail	0	
z Critical one-tail	1.644853	
P(Z<=z) two-tail	0	
z Critical two-tail	1.959961	

Applet vs Message
z-Test: Two Sample for Means

	Variable 1	*Variable 2*
Mean	3.644615	4.318421
Known Variance	12.21	0.23
Observations	39	38
Hypothesized Mean Difference	1	
z	-2.96293	
P(Z<=z) one-tail	0.001524	
z Critical one-tail	1.644853	
P(Z<=z) two-tail	0.003047	
z Critical two-tail	1.959961	

Fig. 21. Z Test for Wireless 9600 b/s client connectivty. The first column contains the first queries comparison and the second column contains the subsequent queries comparison. A similar analysis for the Fixed Network 10 Mb/s and the Dial up 28,800 can be found in [24] or www.ada.cs.ucy.ac.cy/~cssamara/dbms-agents.

expensive networks, such as wireless wide area networks (WANs). Moreover, the framework was shown to be

1. Flexible: It could be set up dynamically and efficiently,
2. Scalable: Its extension to support multidatabase systems not only maintained but also increased its performance benefits, and
3. Robust: The statistical analysis found the DBMS-aglet framework more stable than the current JDBC-based database connectivity.

In addition, the implementation of the framework showed that it outperforms the current approach. In fact, in wireless and dial-up environments and for average size transactions, the client/agent/server adaptation of the framework (namely the "Parked aglet using Message" method) provides a performance improvement of approximately a factor of ten. For the fixed network, the gains are about 40 percent and 30 percent, respectively. Considering that the Aglets-based implementation platform is more tuned towards functionality than performance, the possibi-

lities for even better performance are extremely positive. This assumption is substantiated by early experiments conducted with the Voyager implementation. Finally, the framework is generic and portable and can be used not only within the Web but stand-alone as well for direct Java database connectivity.

APPENDIX A

DESCRIPTIVE STATISTICAL ANALYSIS—SUMMARY

Please see Tables 2, 3, and 4.

APPENDIX B

HISTOGRAMS—WIRELESS 9600

Please see Fig. 20. A similar analysis for the Fixed Network 10 Mb/s and the Dial up 28,800 can be found in [24] or www.ada.cs.ucy.ac.cy/~cssamara/dbms-agents.

APPENDIX C

Z TEST—WIRELESS 9600

The Z-test is used to compare the mean times required by a Web client to query a remote database (for first or subsequent request, given a client connectivity case). The null hypothesis of the Z-tests suggests that the mean time for querying a remote database for the first or subsequent time using the first approach is less than using the second. Comparing the critical with the statistic value, the null hypothesis is either accepted or rejected. By repeating this procedure, as shown in Fig. 21, we conclude that for first queries, the best approach for all cases of client connectivity is the "DBMS-aglet with Messages" and for subsequent queries, the best approach is the "Traditional Applet."

When the queries' cost has been taken into combination, namely for the short and long transaction, the Z test indicated that *the "DBMS-aglet with Messages" is the best approach for all connectivity cases and the applet approach the worst except in the fixed network case.*

ACKNOWLEDGMENTS

The authors would like to thank Hariklea Kazeli for performing the statistical analysis and Constantinos Spyrou for implementing part of this work. We also would like to thank Marios Dikaiakos for his comments and suggestions and Mitsuru Oshima from IBM Tokyo for his valuable assistance in our initial efforts to "master" the Aglets Workbench.

REFERENCES

[1] "The Java™ Language: An Overview, White Paper," available at http://java.sun.com/docs/white/index.html.
[2] E. Pitoura and G. Samaras, *Data Management for Mobile Computing.* Kluwer Academic, 1997.
[3] B. Jepson, *Database Connectivity: The Lure of Java, Java Report.* Whiley Computer Publishing, 1997.
[4] B. Jepson, *Java Database Programming.* Wiley Computer Publishing, 1997.
[5] R. Greem, *Article: Java Access to SQL Databases.* Canadian Mind Products, 1997.
[6] C.G. Harrison, D.M. Chessm, and A. Kershenbaum, "Mobile Agents: Are They a Good Idea?," research report, IBM Research Division, 1995.
[7] IBM Japan Research Group "Aglets Workbench,"web site: http://aglets.trl.ibm.co.jp.
[8] E. Anuff, *Java Sourcebook.* Whiley Computer Publishing, 1996.
[9] B.F. Burton and V.W. Marek, *Applications of Java Programming Language to Databases Management.* Univ. of Kentucky, 1997.
[10] T. Berners-Lee and D. Connolly, "Hypertext Markup Language Specification 2.0," internet draft, Internet Eng. Task Force (IETF), HTML Working Group, available at www.ics.uci.edu/ietf/html/html2spec.ps.gz, June 1995.
[11] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol—HTTP/1.0 Specification," internet draft, Internet Eng. Task Force (IETF). Available at www.ics.uci.edu/pub/ietf/http/draft-fielding-http-spec-01.ps.Z, Aug. 1995.
[12] T. Berners-Lee, R. Caililiau, A. Luotonen, H.F. Nielsen, and A. Secret, "The World Wide Web," *Comm. ACM,* vol. 37, no 8, pp. 76-82, Aug. 1994.
[13] Sun Microsystems Inc. *Java Development Kit,* web site http://java.sun.com.
[14] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik, "Itinerant Agents for Mobile Computing," *J. IEEE Personal Comm.,* vol. 2, no. 5, Oct. 1993.
[15] J.E. White, "Mobile Agents. General Magic White Paper," web site http://www.genmagic.com/agents, 1996.
[16] Z. P. Lazar and P. Holfelder, "Web Database Connectivity with Scripting Languages," *World Wide Web J.,* web site: http://www.w3j.com/6/s3.lazar.html, Spring 1997.
[17] S.P. Hadjiefthymiades and D.I. Martakos, "A Generic Framework for the Development of Structured Databases on the World Wide Web," *Proc. Fifth Int'l World Wide Web Conf.,* web site http://www.w3.org/, May 1996.
[18] *Borland DataGateway for Java,* available at http://www.borland.com/datagateway/.
[19] "Symantec dbANYWHERE,"available at http://www.symantec.com.
[20] "IBM DB2,"available at http://www.software.ibm.com/data/db2/.
[21] G. Samaras, E. Pitoura, and P. Evripidou, "Software Models for Wireless and Mobile Computing: Survey and Case Study," Technical Report TR-99-5, Univ. of Cyprus, Mar. 1999.
[22] ObjectSpace Voyager,™ "Technical Overview,"web site: http://www.objectspace.com/voyager/whitepapers/VoyagerTechOview.pdf.
[23] J.C. Daniel and W.W. Terrell, *Business Statistics for Management and Economics.* seventh ed., 1995.
[24] S. Papastavrou, E. Pitoura, and G. Samaras, "Mobile Agents for WWW Distributed Database Access," (Extended version) Technical Report TR 98-12, Univ. Of Cyprus, Computer Science Dept., Sept. 1998.
[25] M. Breugst, I. Busse, S. Covaci, and T. Magedanz, "Grasshopper: A Mobile Agent Platform for IN Based Service Environments," *Proc. IEEE IN Workshop,* May 1998.
[26] J. White, "General Magic White Paper," http://www.genmagic.com/agents/, 1996.
[27] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet, " Concordia: An Infrastructure for Collaborating Mobile Agents," *Lecture Notes in Computer Science,* vol. 1,219, 1997. http://www.meitca.com/HSL/Projects/Concordia/.
[28] D. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets.* Addison Wesley, 1998.
[29] T. Papaioannou et. al., "Mobile Agents Technology in Support of Sales Order Processing in the Virtual Enterprise," *Proc. Third IEEE/IFIP Int'l Conf. Information Technology for Balanced Automation Systems in Manufacturing,* L.M. Camrinha-Matos, H. Afsaranesh, and V. Marik, eds., p. 23, Aug. 1998.
[30] S. Weissman Lauzac and P.K. Chrysanthis, "Programming Views for Mobile Database Clients," *Proc. Ninth DEXA Conf. and Workshop Database and Expert Systems Applications: Mobility in Databases and Distributed Systems,* Aug. 1998.
[31] G. Samaras, M. Dikaiakos, C. Spyrou, and A. Liberdos, "Mobile Agent Platforms for Web-Databases: A Qualitative and Quantitative Assessment," *Proc. the Joint Symp. (ASA/MA '99), Proc. First Int'l Symp. Agent Systems and Applications (ASA '99), Proc. Third Int'l Symp. Mobile Agents (MA '99),* pp. 50-64, 1999.

[32] M. Dikaiakos and G. Samaras, "A Performance Analysis Framework for Mobile-Agent Systems," *Proc. First Ann. Workshop Infrastructure for Scalable Multi-Agent Systems, Proc. Fourth Int'l Conf. Autonomous Agents 2000,* June 2000.
[33] S. Papastavrou, P.K. Chrysanthis, G. Samaras, and E. Pitoura, "An Evaluation of the Java-based Approaches for Web Database Access," *Proc. of the Fifth IFCIS Int'l Conf. Cooperative Information Systems (CoopIS '2000),* Sept. 2000.
[34] Oracle, http://www.oracle.com/oramag/oracle/00-Jan/10m.ob.htm and http://www.oracle.com/mobile/.
[35] Allaire "Cold Fussion,"available at http://www.allaire.com/products/coldfusion/index.cfm.
[36] G. Helmayer, G. Kappel, and S. Reich, "Connecting Databases on the Web: A Taxonomy of Gateways," *Proc. Eighth DEXA Int'l Conf. and Workshops,* Sept. 1997.

Stavros Papastavrou received the BSc degree from the Department of Computer Science of the University of Cyprus, Cyprus in 1998 and the MSc degree in computer science from the University of Pittsburgh in 2000. He is now a PhD canndidate at the University of Pittsburgh. His work on utilizing mobile agents for Web database access has received the best paper award of the 1999 IEEE International Conference on Data Engineering (ICDE '99). His research interests include mobile agents technology, Web databse connectivity, and mobile data management. He is a student member of the IEEE

George Samaras received the PhD degree in computer science from Rensselaer Polytechnic Institute in 1989. He is currently an associate professor at the University of Cyprus, Cyprus. He was previously at IBM Research Triangle Park, North Carolina, and taught at the University of North Carolina at Chapel Hill (adjunct assistant professor, 1990-1993). He served as the lead architect of IBM's distributed commit architecture (1990-1994) and coauthored the final publication of the architecture (IBM Book, Sc31-8134-00). He was a member of IBM's wireless division and particpated in the design/architecture of IBM's Web*Express*, a wireless Web browsing system. He recently (1997) coauthored a book on data management for mobile computing (Kluwer Academic). He has a number of patents relating to transaction processing technology and numerous technical conference and journal publications. His work on utilizing mobile agents for Web databse access has received the best paper award of the 1999 IEEE International Conference on Data Engineering (ICDE '99). He has served as proposal evaluator at a national and international level and he is regularly invited by the European Commission to serve as project evaluator and auditor in areas related to mobile computing and mobile agents. He also served on IBM's internal international standards committees for issues related to distributed transaction processing (OSI/TP, XOPEN, OMG). His research interests includes mobile computing, mobile agents, transaction processing, commit protocols and resource recovery, and real-time systems. He is a voting member of the ACM and the IEEE Computer Society.

Evaggelia Pitoura received the Bsc degree from the Department of Computer Science and Engineering of the University of Patras, Greece in 1990 and the MSc and PhD degrees in computer science from Purdue University in 1993 and 1995, respectively. Since September 1995, she has been on the faculty of the Department of Computer Science of the University of Ioannina, Greece. Her main research interest are data management for mobile computing and network-centric databases. Her publications include several articles in international journals (including *IEEE Transaction on Knowledge and Data Engineering*, *ACM Computing Surveys*, *Information Systems*) and conferences (including VLDB, ICDE, ICDCS, CIKM) and a recent published book on mobile computing. She received the best paper award in the IEEE International Conference on Data Engineering (ICDE '99) for her work on mobile computing agents. She has also coauthored a tutorial on mobile computing presented in IEEE International Conference on Data Engineering (ICDE '00). She has also served on a number of program committees and was program cochair of the MobiDE workshop held in conjunction with MobiCom '99. She is a member of the IEEE.