Mobile Agents for WWW Distributed Database Access

Stavros Papastavrou
Department of Computer Science
University of Cyprus
CY-1678 Nicosia, Cyprus

George Samaras
Department of Computer Science
University of Cyprus
CY-1678 Nicosia, Cyprus
cssamara@turing.cs.ucy.ac.cy

Evaggelia Pitoura
Department of Computer Science
University of Ioannina
GR 45110 Ioannina, Greece
pitoura@cs.uoi.gr

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 methodologies 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 light-weight, portable and autonomous clients as well as operation on slow or expensive networks. The implementation of the framework shows that its performance is comparable to, and in some case outperforms, the current approach. In fact, in a wireless and dial-up environments and for average size transactions, a client/agent/server adaptation of the framework provided a performance improvement of approximately a factor of ten. For the fixed network the gains were about 40% and 30% respectively.

1. Introduction

The widespread use of Java[1] [1] in network-centric computing, attributed mainly to its global 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 has 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 Web suggests that Web browsers may offer a compelling end-user interface for a large class of applications including database management systems (DBMSs).

The objective is to decisively marry these two technologies, namely Java and Web, for the retrieval of information residing in relational databases. The real challenge is the formation of smart, lightweight, flexible, independent and portable Java DBMS client programs, with maximum performance and security standards, that will support database connectivity over the Internet. However, the currently proposed methodologies [5] overload the client 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 the idea of 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 methodologies.

The proposed framework, called the "DBMS-Aglet Framework" utilizes the newest 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 which cooperate to efficiently support Web database connectivity. The main agent, called DBMS-aglet, is database capable with the capabilities dynamically acquired 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 more essential tasks.

This 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 light-weight, portable and autonomous clients as well as operation on slow or expensive networks. The implementation of the framework shows that its performance is comparable to, and in some cases outperforms, the current approaches. In fact, in 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% and 30% respectively. This framework is generic and portable and it can be used not only within the Web but stand-alone as well for direct Java database connectivity.

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. Performance evaluation is presented in Section 5. In Section 7, we present future work. Section 8 concludes the paper.

[1] Java is a trademark of Sun Microsystems.

2. Background Material

2.1 Supporting Technologies

2.1.2 Java and JDBC. Java is an object-oriented, interpreted, robust, secure, architecture neutral, portable, and multi-threaded language. The uniqueness of Java lies on the fact that it combines both compiled and interpreted code. The Java executable code (called bytecode) is generated by platform-depended compilers and runs on any platform as long as its operating system is running the Java Interpreter or any Java enabled web browser. In fact, 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 due to 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 which provides a Java interface to the relational database, and the JDBC driver API which executes/implements this interface. A client that employees the JDBC API must first download to its environment a JDBC driver before accessing a particular database.

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, and 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 so far obtained data. 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 giving the aglet the sense of self-governing and the look of an intelligent agent.

An aglet can be dispatched to any remote host that supports the Java Virtual Machine. This requires from the remote host to pre-install 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, the IBM Aglet team provided the so-called "FijiApplet", an abstract applet class that is 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 from and retracted back to the FijiApplet. For a Java-enabled web browser to host and fire aglets to various destinations two more components are required and are provided by the Aglet's Framework. These are an aglet plug-in to allow the browser to host aglets and an aglet router that must be install at the Web server to capture incoming aglets and forward them to their destination.

2.2 The Current Approach: Applets for Distributed Database Access

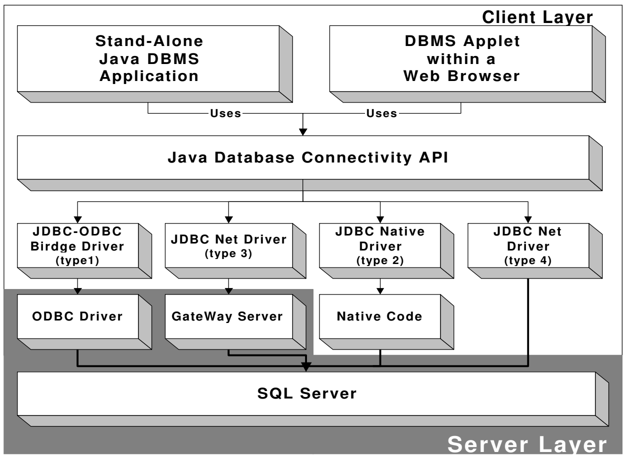

Figure 1: Standard JDBC Methodologies

The challenge is to provide the Internet user with Web-based database connectivity with the lower possible overhead. Various approaches have been proposed and used [16, 17]. However, the concept of running applets within Java enabled browser that utilize the JDBC application interface is increasingly gaining popularity with the major database vendors. Such ``database'' applets function similarly to an independent, powerful and intelligent database web client. There are four ways a Java applet can gain access to a remote relational database (Figure 1); making use of one of the four existing types of JDBC driver implementation.

The first approach makes use of the JDBC-ODBC bridge driver (that is the JDBC driver type 1). What the JDBC-ODBC bridge does is to translate the JDBC API calls into ODBC[2] calls

[2] Open Database Connectivity: An API that defines the routines for accessing MS-windows Databases.

and send 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 pre-installed on the client machine. This approach poses many extra layers of overhead and thus, it can not be seriously considered for serving the public on the web. Nevertheless, this approach is useful in accessing Windows based 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[3] code. The client applet, through this JDBC driver (type 2 driver), speaks directly the protocol of the remote SQL database. In spite of the ultimate speed with which an applet accesses a remote database, this method suffers from requiring the client to pre-install native code.

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 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].

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

Table 1: The Web-based DBMS-aglet Infrastructure

Last, but not least, the fourth approach makes use of a JDBC 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. Furthermore, by utilizing the JDBC API classes at the client machine the client tends to be transformed from a lightweight client to a complete conventional, data-aware, LAN client with full functionality. Our primary concern is to simplify and relief the remote client. What a remote database client, running within a web browser, needs to handle is not a complex set of JDBC interface classes, but only the input to be passed for a request, and the expected formatting of the output, so it can represent it using GUI Java components. We consider it preferable, for lightweight clients (especially wireless ones), to spend resources in the GUI representation of data, rather than loading the client with vain capabilities.

[3] Native code: Database-Vendor specific code.

3. DBMS-Aglets: Mobile Agents for Distributed Database Access

Motivated by the promising technology of mobile agents, we suggest a totally different approach for Web-based database access than the existing conventional and commercial implementations. Our approach offers a more efficient and anthropomorphic framework for Web-oriented, DBMS client/server computing that avoids the pitfalls and overheads of applet-based solutions described earlier.

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 host agents with database capabilities. In addition, the existing aglets need to be extended to be database capable and supporting aglets need to be provided to aid such database-capable aglets in their negotiations with the SQL server. Specifically, the following components are needed to support the DBMS aglet framework:

- 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, also an extension of the Aglet class, is a permanent resident of an 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.

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 that provides the facilities for hosting and routing mobile agents. The mobile agent infrastructure used in our implementation is aglets. 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 (a) the DBMS-aglets framework, (b) the aglets framework, and (c) the standard networking infrastructure.

3.2 Demonstrating the Web-based DBMS-Aglet Framework

To demonstrate our methodology, 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 in 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.

Figure 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]). It is up to the programmer's skills to design the User Interface (UI) part of the DBMS-applet to form a convincing DBMS client 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 takes messages- directions from its creator DBMS-applet corresponding to the nature of the user's query. Those 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 source) in order to satisfy the request of the DBMS-applet user. 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 (if the user originally directed it so), 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 & 5).

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

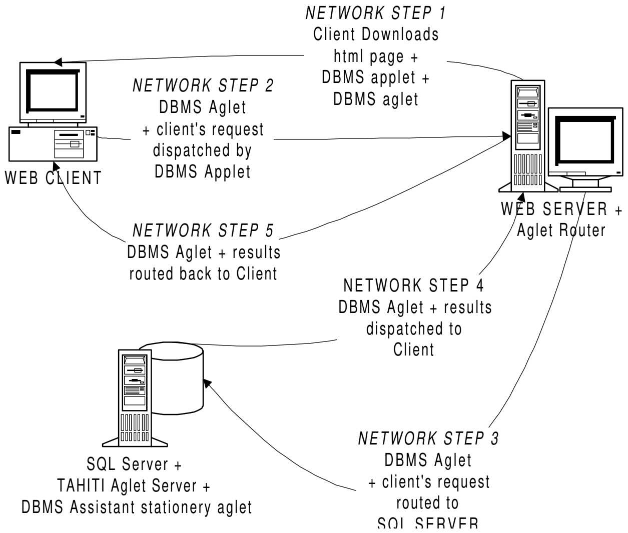

Figure 2: The DBMS-Aglets Life-cycle

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 any JDBC drivers. This downloading of the JDBC driver and connection 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.

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 [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 it is 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 Figure 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 identical 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.

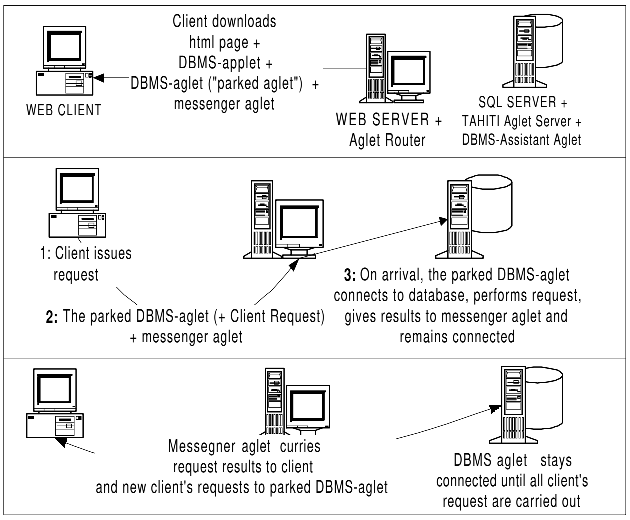

Figure 3: DBMS-aglet Framework Using a Messenger Aglet

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 to the one way and new queries to the other.

This overhead can be eliminated by replacing the messenger aglet with two type of messages. The first kind of 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 methodology demonstrates a true service-specific client/agent/server application. The aglet is literally inserted into the path between the client and server communicating with each other via messages.

Thus, by using a parked DBMS-aglet, we avoid the reconnection cost (just like with the messenger approach) and by using messages instead of the messenger aglet we eliminate the time of negotiation and the amount of the transmitted data between the client and the server. As we will see later on 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 its itinerary. The rest is the responsibility of the DBMS-aglet. The effect on performance is quite significant, this delegation of responsibility resulted 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 can not (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 encounter obstacles, like 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, cooperate and communicate with each other 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 multiple DBMS-aglets. Only the final result is reported to the DBMS-applet. This approach is more advantageous in that it 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. 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 class names of all the different available DBMS-aglets that can be created.
- Maintain a list of aglet proxies, aglet Ids, and locations of the DBMS-aglets.
- Maintain a list of all the possible workstationYs URLs that can host DBMS-aglets.
- Maintain a list of all the workstationYs 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 (Figure 4). The target workstations can be either computers in the fixed local network or computers connected through the Internet.

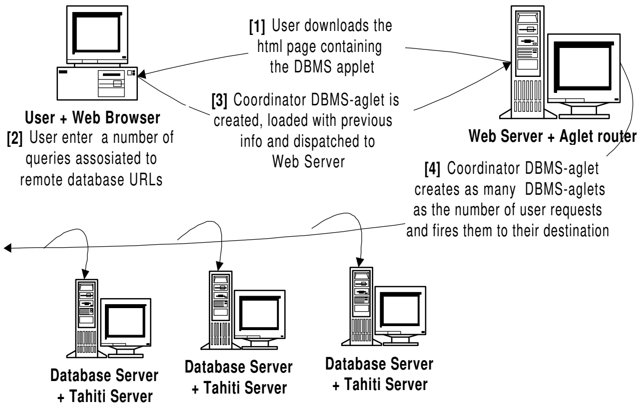

Figure 4: The DBMS-Aglet Multidatabase Framework

4.2 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 change 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, more DBMS-aglets can be created as needed. The databases that will host the DBMS-aglets can be also decided 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 approach, the client must download a large number of JDBC drivers; one for each remote database to be accessed. This is very costly, extremely 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. 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 methodologies. In more detail, we are interested in the time required, for each methodology, to query the remote database for the first time and for any subsequent requests. We consider both short (three queries) and long transactions (six queries) between the client and the remote database. Short or long transactions are the type of transactions generally anticipated by Web users.

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

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

For each methodology 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 AM and 5 PM. Each tested methodology provides two data sets of results (observations); one for the first query and one for the subsequent queries.

5.1 The DBMS-Aglets

Due to availability reasons we only used a Borland provided JDBC type 3 driver[4] and perform 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 is quite simple including a Web server, a remote relational database (namely Borland's Paradox) and a Windows 95 client. Furthermore, the degree of distribution employed was a bit limited confined only within the boundaries of a small geographic area.

This, however, seems to benefit the applet approach. For the fixed network case, however, we performed the test within the large University network.

For the applet-based approach, we (a) installed at the Web server a type 3 JDBC driver, (b) created an applet responsible for accessing and querying the remote database, called db-applet, and (c) 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 included 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 included 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 currying 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 received 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.

5.1.2 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 methodology, 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 methodology for all cases of client connectivity. Figures x, y, z shows the mean time required for all methodologies.

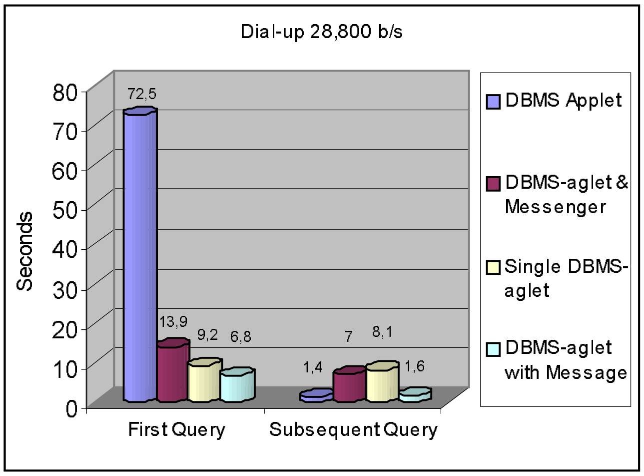

Figure x: Mean times for 28,800 b/s client connectivity

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 methodology. For example, as shown in Figure h1 and h2 (histograms) with a client connectivity of 9,600b/s for the first query, the spread of the time required for the traditional DBMS applet is quite wide having a variance value of 612,34, where as for the 'DBMS-aglet with Messages' methodology the spread is quite limited with a variance of 0,652. This wide spread in the Applet methodology 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

[4] The driver is called borland.jdbc.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 borland.jdbc.Bridge.LocalDriver, it translates the JDBC API into paradox SQL commands.

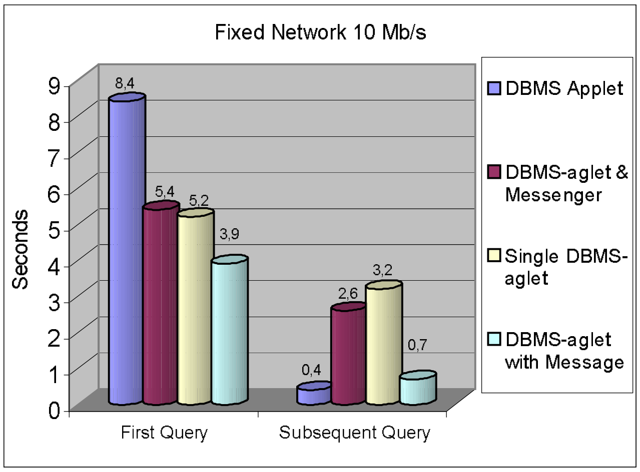

Figure y: Mean times for 10Mb/s client connectivity

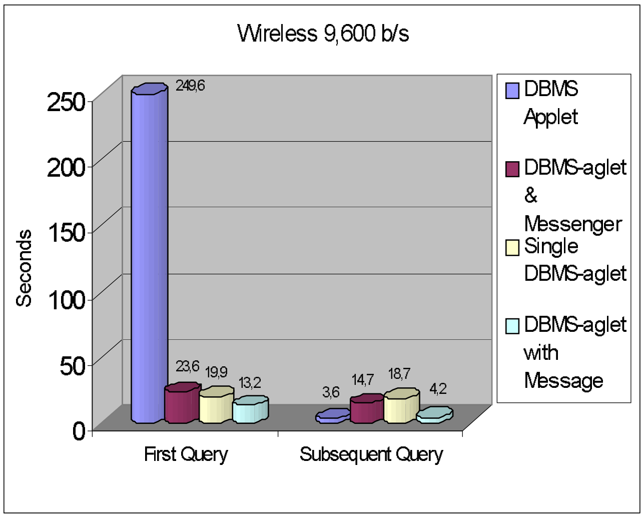

Figure z: Mean times for 9,600 b/s client connectivity

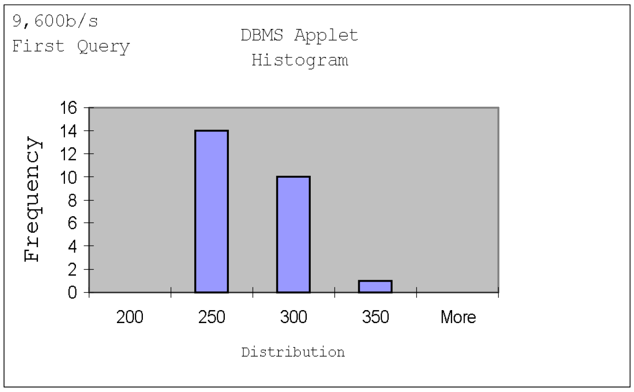

Figure h1: Distribution in Traditional Applet methodology, first query

Messages' methodology 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 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 methodologies.

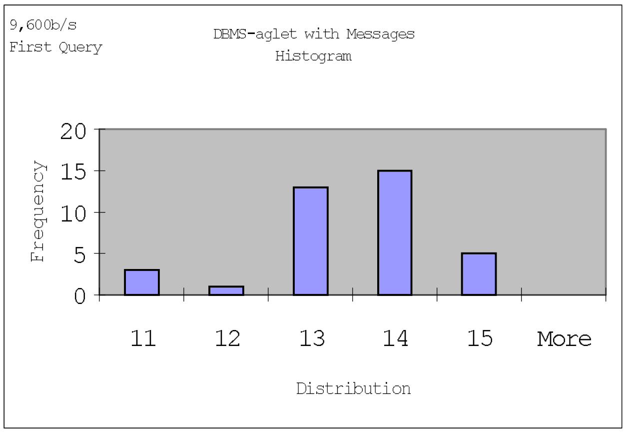

Figure h2: Distribution in the DBMS-aglet with Messages methodology, first query

The reliability of the mean as a measure of tendency for each sample and the lack of any linear dependencies among the methodologies, 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 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 methodology, either for first or for subsequent query, with a certain client connectivity.

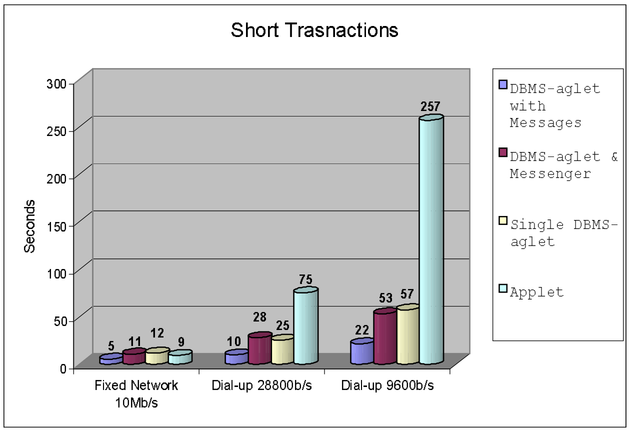

Figure t1: Short Transactions Graph

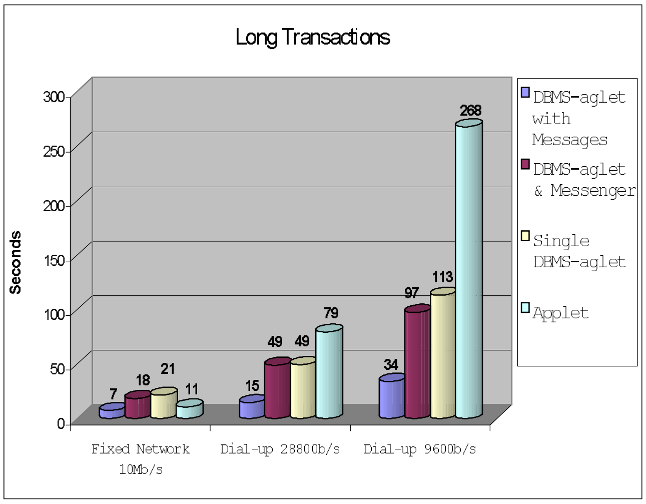

Figure t2: Long Transactions Graph

The Z test proved that the time required for the "DBMS-aglet with Messages" methodology 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 methodologies concerning subsequent queries proving that the best methodology for all cases of client connectivity is the "traditional Applet". Taking the results from the Z test, we safely produced the time required for short and long transaction for each methodology under a given Web client connectivity case. A short transaction's cost is the sum of (a) the time required for the first query and (b) the time required for two subsequent requests. For a long transaction the b factor is the time required for five subsequent requests. Figure t1 and t2 illustrate the time required for short and long transactions respectively.

As shown in both graphs, the 'DBMS-aglet with Messages' methodology requires considerably less time than any other methodology, and the applet methodology 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% and 30% respectively.

5.2.1 Prototype Implementation: Accessing Multiple Databases. To demonstrate the DBMS-aglet multidatabase framework, we have setup a Mobile infrastructure network and created a simple application. 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 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 sub-queries and the IP addresses of various hosts where the databases are located. When the user enter 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, e.t.c.
- 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 delivered the results to the Coordinator DBMS-aglet. The Coordinator DBMS-aglet using a Java-thread based approach joint the various results and delivered the final outcome to the client.

5.2.2 Performance Comparison. We have performed a similar to the single DBMS-aglet approach statistical analysis. The Multidatabase DBMS-aglet approach, as shown in figure 5, by far outperforms the traditional applet approach. Figure 5, 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 5.2.1), completely outstrips the traditional applet methodology.

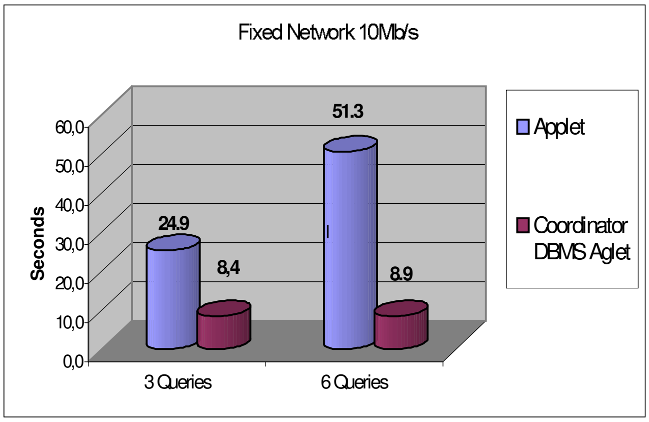

Figure 5: Multidatabase DBMS-Aglets Vs Traditional Multidatabase Applets

The multidatabase aglet approach performs extremely well since it does not have to deal with the downloading of the various JDBC drivers and its dealings with the client are minimized to only the final result. The performance between the two configurations does not significantly changes since the DBMS aglets are send to the various destinations to work in parallel. For the reverse reasons, the applet approach shows a linear degradation in performance.

6. Discussion and Future Work

Web technology advanced distributed systems by providing the underlying fabric for communication among participating process located at remote computers connected by the internet. These modern network-centric distributed systems rely on two emerging architectures/technologies; the Common Request Broker Architecture (CORBA) and mobile agents, whose goals are at this point complementary. CORBA provides for location and language transparency, while mobile agents extend the communication paradigm by providing for mobility, independence, autonomy and intelligence of code and data. We expect that systems, such as VOYAGER [22], that incorporate both CORBA and mobile agent features, will further extend the gains offered by this work.

A Java mobile execution environment extends the scoping 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 simply 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 view, a DBMS-CacheAglet, a DBMS-PreprocessAglet or a DBMS-ParallelAglet with parallel processing capabilities. Figure 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.

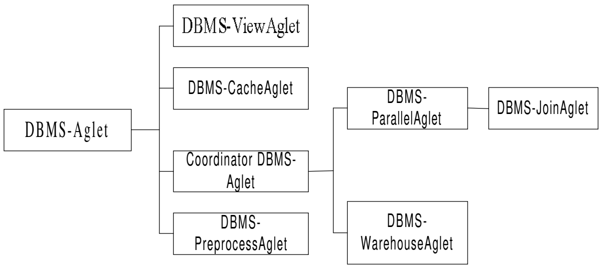

Figure 6: This is an example of the visionary library

7. Conclusions

In this paper, we have introduced a new approach for developing client/server applications on 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 expensive networks, such as wireless wide area networks (WANs). Moreover, the framework was shown to be (a) flexible, (b) scalable: its extension to support multidatabase systems not only maintained but also increased its performance benefits, and (c) 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) provided a performance improvement of approximately a factor of ten. For the fixed network the gains were about 40% and 30% respectively. 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.

Acknowledgments. The authors wish to thank Hariklea Kazeli for performing the statistical analysis and Spyrou Konstantinos for his so needed contribution in the implementation part of this work. We also like to thank Mitsuru Oshima from IBM Tokyo for his valuable assistance in our initial efforts to ‘master’ the Aglets Workbench.

References

[1] G. Cornel and C. Horstman. Core Java. SunSoft Press, 1996.

[2] Evaggelia Pitoura and George Samaras. Data Management for Mobile Computing. Kluwer Academic Publishers, 1997

[3] B. Jepson. Database Connectivity: The Lure of Java, Java Report. Whiley Computer Publishing, 1997.

[4] B. Jepson. Java Database Programming. Wiley Computer Publ., 1997.

[5] Roedy Greem. Article: Java Access to SQL Databases. Canadian Mind Products, 1997.

[6] Colin G. Harrison, David M. Chessm, Aaron Kershenbaum. Mobile Agents: are they a good idea? Research Report, IBM Research Division.

[7] Aglets Workbench, by IBM Japan Research Group. Web site: <http:/ aglets.trl.ibm.co.jp>

[8] Ed Anuff. Java sourcebook. Whiley Computer Publishing, 1996.

[9] Bradley F. Burton and Victor W. Marek. Applications of Java programming language to databases management. University of Kentucky, 1997.

[10] T. Berners-Lee and D. Connolly, Hypertext Markup Language Specification 2.0, Internet Draft, Internet Engineering 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, H. Frystyk. Hypertext Transfer Protocol - HTTP/1.0 Specification, Internet Draft, Internet Engineering Task Force (IETF). Available at <www.ics.uci.edu/pub/ietf/http/draft-fielding-http-spec-01.ps.Z>, August 1995.

[12] T. Berners-Lee and R. Caililiau and A. Luotonen and H.F. Nielsen and A. Secret, The World Wide Web, Journal Communications of the ACM, Vol. 37, No 8, pp. 76-82, August, 1994.

[13] Java Development Kit, by Sun Microsystems Inc. Web site <http://java.sun.com>

[14] D. Chess, B. Grosof, C. Harrison, D.Levine, C. Parris, G. Tsudik. Itinerant Agents for Mobile Computing. Journal IEEE Personal Communications, Vol. 2, No. 5, October 1993.

[15] J. E. White. Mobile Agents. General Magic White Paper. Web site <http://www.genmagic.com/agents>, 1996.

[16] Z. Peter Lazar and Peter Holfelder. Web Database Connectivity with Scripting Languages. World Wide Web Journal. Web site <http://www.w3j.com/6/s3.lazar.html>, Spring 1997.

[17] Stathes P. Hadjiefthymiades and Drakoulis I. Martakos. A generic framework for the development of structured databases on the World Wide Web. Fifth International World Wide Web Conference. 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] George Samaras and Evaggelia Pitoura. Computational Models for the Wireless and Mobile Environments. Technical Report TR-98-4, University of Cyprus, Computer Science Department.

[22] ObjectSpace Voyager [tm] 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, 7th Edition, 1995, ISBN-0395712319.

[24] S. Papastavrou, E. Pitoura, G. Samaras. Mobile Agents for WWW Distributed Database Access. (extended version). Technical Report TR 98-12, Univ. Of Cyprus, Computer Science Department, Sept. 98.