A Survey on the Java-based Approaches for Web Database Connectivity[1]

Stavros Papastavrou[2], Panos K. Chrysanthis[2], George Samaras[3], Evaggelia Pitoura[4]

Abstract

The undeniable popularity of the web makes the efficient accessing of distributed databases from web clients an important topic. Various methods for web database integration have been proposed but recently there is an increasing interest on those based on Java-based ones. This is due to the inherent advantages of Java, which supports platform independence and secure program execution, and produces a small size of compiled code. In this experimental paper, we evaluate all currently available Java-based approaches. These include Java applets, Java Sockets, Servlets, Remote Method Invocation, CORBA, and Mobile Agents. To this end, we implemented a Web client accessing a remote database using each of these approaches and compared their behavior along the following important parameters: (1) *performance* expressed in terms of response time under different loads, (2) *transparency of communication* expressed in terms of complexity of networking API, and (3) *extensibility expressed* in terms of ease of adding new components.

Keywords: **Distributed Databases, WWW, Java Mobile Agents, Distributed Objects, CORBA**

I. INTRODUCTION

Providing efficient access to distributed databases from Web clients using a Web browser [2] is crucial for the emerging database applications such as E-Commerce. Several methods for Web database connectivity and integration have been proposed such as CGI scripts, active servers pages, server side include, databases speaking http, external viewers or plug-ins, proxy-based, and HyperWave [4]. However, there is an increasing interest in those that are Java-based due to the inherent advantages of Java [1], namely, platform independence support, secure program execution, and production of a small size of compiled code.

Several Java-based methods are currently available for Web database integration but in the best of our knowledge, there is no quantitative comparison of them. This experimental paper contributes such a comparison. Specifically, it evaluates six approaches, namely, *Java JDBC applet, Java Sockets* [7], *Servlet* [7], *Remote Method Invocation (RMI)* [7], *CORBA* [9], and *Java mobile agents (JMA)* [3]. Each approach differs in the way the client establishes connection with remote database servers.

For our evaluation, we used each of these approaches to implement a Web client accessing a remote database and compared their behavior along the following important parameters: (1) *performance* expressed in terms of response time under different loads, (2) *transparency of communication* expressed in terms of complexity of networking API, and (3) *extensibility* expressed in terms of ease of adding new components. Further, we characterized these approaches in terms of the total development effort based on lines of code at both the client and the server side in conjunction with the two latter parameters, namely, transparency and extensibility.

In the next section, we briefly describe our testbed. In Sections III to IV, we elaborate on the characteristics of each approach when comparing them along the dimensions of communication transparency and extensibility. In section VI, we present our performance evaluation results.

II. EXPERIMENTAL TESTBED

Two design principles were adopted in the selection of the various components during the development of the testbed. First, our Web clients should be lean with the purpose of allowing fast downloads and therefore increasing support for wireless and mobile clients. Second, no a-priori configuration of the Web client should be necessary to run the experiments in order to maintain portability, and therefore support arbitrary clients.

For every approach, our Web client program was a Java applet, which was installed on a Web server machine along with an html page. Every experiment was initiated by first pointing to the html page from a remote client computer (Figure 1). After the Java applet was initialized at the client computer, queries were issued through the applet's GUI and executed at the remote database server. Our remote database server, a 3-table relational Microsoft Access database, was installed on the same machine with the Web server. The communication between the server and the client computer was a wireless LAN at 1.2Mbps.

In all cases, the client establishes Web database connectivity through a middleware program typically running on the Web server machine. For the Java JDBC applet, we used a type 3 JDBC driver in accordance to our design criteria. In this case, the middleware corresponds to the middle-tier gateway of the type 3 JDBC driver [8]. In the case of JMA, the middleware is a local stationary agent that provides the information necessary for a mobile agent to load the appropriate JDBC driver and connect to the database server. In our experiments, we used DBMS-aglets [6]. In all other cases, we developed the middleware program, which plays the role of an application server that uses a JDBC-ODBC bridge driver to connect to the database server. For more details, see [5].

[1] This work was partially supported by NSF grants IRI-9502091 and IIS-9812532, and AFOSR award F49620-98-1-043.

[2] Computer Science Dept., Univ. of Pittsburgh, PA 15260, USA.

[3] Computer Science Dept., Univ. of Cyprus, Nicosia, Cyprus.

[4] Computer Science Dept., Univ. of Ioannina, Ioannina, Greece.

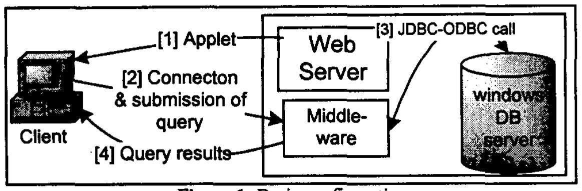


Figure 1: Basic configuration

III. Transparency of Communication

The dimension of the transparency of communication deals with the level of abstraction of communication between the client and the server side, in other words, between the client and the middleware program. The approaches can be broadly classified as (1) *non-RPC* ones, that do not support any clear remote method invocation mechanism, and (2) *RPC* ones with clear remote method invocation semantics. The Java Sockets and Servlets are non-RPC approaches in which information between the client and the middleware program is exchanged using streams of data. Java JDDC applets, RMI, CORBA and JMA are all RPC approaches.

Table 1 compares the transparency of communications of the discussed approaches based on the complexity of the networking API employed by each approach. Clearly, the RPC approaches involve less complex networking APIs and hence more transparent client/server communication.

The CORBA approach offers the *highest* level of communication transparency since it requires knowledge of neither URLs nor port numbers to establish database connectivity. It only requires the reference name with which the application server was registered at the server site. One level lower (*high*) is the RMI approach, which requires the URL of the database server along with the reference name of the application server. Similar to the RMI approach, the Java JDBC Applet approach requires the URL of the database server, and a data source name, which identifies the database itself. In this same level is the JMA approach. Mobile agents identify remote host machines with their URL, and interact with other agents using their unique identifiers. One level below (*low*) is the Servlet approach requiring a URL, the servlet name, and the type of operation to be executed by the particular servlet. Finally, as expected, the approach with the *lowest* communication transparency is the socket approach, which requires knowledge of the IP and of the port number of the application server.

IV. Extensibility

We define *extensibility* to be: (a) the ability of adding new components to an approach (e.g., a new application server object attached to a local or a remote database) and binding them with the existing ones at the server site; and (b) the level of modifications needed at the client part that will enable the client to utilize newly added components. We classified the various approach in terms of extensibility as highest, high, average and low (Table 1).

The approaches with the *highest* degree of extensibility are the CORBA and JMA. In the one based on CORBA, the application server and the client applet can bind to a newly added component by only using its reference name. As opposed to other approaches, new components need not be necessarily located at the Web server machine in order for the client to bind to them.

	Applet JDBC	Socket	Servlet	RMI	CORBA	JMA
Transparency	High	Lowest	Low	High	Highest	High
Extensibility	High	Lowest	High	Average	Highest	Highest
Code Size	4c	22c	16c	9c	10c	30c
Total effort	Lowest	Highest	High	Low	Lowest	Low

Table 1: Effort of development

The JMA approach is inherently very extensible since mobile agents were designed to autonomously collect information and exploit any newly added servers in order to complete their execution plan. Moreover, the Web client need not be aware of the existence of new servers.

The *high* extensibility of the JDBC applet approach is due to the type 3 JDBC driver used. Of all the JDBC drivers, type 3 drivers are the most extensible because of their middle-tier gateway that maps client applet's database requests to any local or remote database calls. The Web client only needs to name the newly added databases.

The servlet approach also offers *high* extensibility. Servlets execute in the context of the Web server and can call (explicitly) other servlets within the same context. This means new servlets can be added without any Web client modification. The client applet can also call explicitly a new servlet using as a reference the URL, the new servlet's name and the type of operation that must be executed by it.

Because of their similarities, one might have expected that RMI and CORBA approaches would exhibit the same degree of extensibility. However, compared to the CORBA approach, the RMI approach is much less extensible for three reasons. First, new components must be written only in Java. Second, new components are identified, besides of their reference name, with an additional URL. Lastly, the client applet cannot bind to new components that reside on URLs other than the Web server. Clearly, RMI also offers lower extensibility than the Java JDBC applet approach.

Finally, the approach with the *lowest* degree of extensibility is the socket one. For any new component, a new socket must be created, bound and managed either at the side of the application server or at the Web client.

V. Effort of Development

The effort of development basically combines the dimensions of transparency of communications and extensibility, and quantifies them in terms of lines of code. In Table 1, the lines of code for each approach are normalized with a constant *c*.

The approaches with the lowest effort of programming are the Java JDBC applet and the CORBA. The applet approach combines the fewer relative lines of code, high level of network transparency and an average extensibility, while the CORBA approach offers the highest transparency and extensibility with relative small code size.

The high extensibility and transparency of the JMA approach comes with a premium in terms of lines of code. Mobile agents involved significant programming. The RMI approach is the opposite of the JMA one. It requires a relatively low number of lines of code and offers average extensibility.

The Servlet and Socket approaches involve the most effort of development, given their large number of lines of code and their low transparency and extensibility.

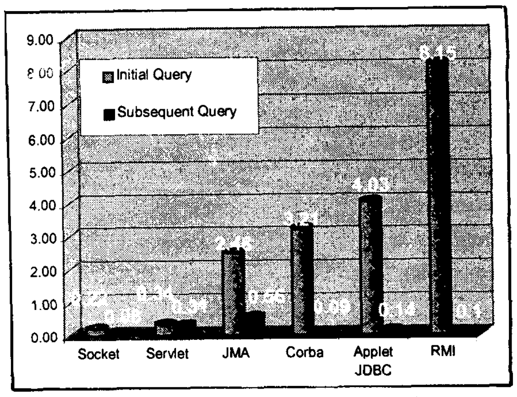
VI. Performance Evaluation

For our performance evaluation, we measured the average response time for our Web client (a) to query the remote database for the first time, (b) to query the remote database for a number of subsequent times. Querying the remote database for the first time differs from subsequent queries because the first query involves the additional overhead of establishing the communication link between the client and the remote database.

The other significant issue that we considered in our experiments is the size of the query result. Query result size directly affects the response time in two ways. First, in the amount of time spent for the query to execute, and second, in the transport time for the results to reach the client. We adjusted the size of the query result by changing the complexity of the SQL statement issued through the client applet. For our experiments we measured average response times for a wide range of query result sizes, beginning from 128 bytes (8 tuples) up to 64 kilobytes (1000 tuples). For each approach, a sufficient number of runs were performed in order to obtain statistically significant results [11]. Below we first focus on the experiments for small query results (128 bytes) and then discuss how the response time of the different approaches is affected by the query size.

Graph 1 shows the average response time for the initial and subsequent queries in each approach. For the initial query, the non-RPC approaches have by far the lowest response time. This can be explained by the fact that their initialization phase does not engage any special package loading or handling by the client. Compared to the Socket approach, the Servlet approach performs slightly worse because (a) the communication between the client and the servlet is marshaled by the Web server, and (b) by executing as a Web server thread, the servlet receives less CPU time than the socket application server. Thus, servlets respond slower to requests and require more time to assemble and return the query results.

From the RPC approaches, the JMA approach offers the best performance for a single (initial) query. Significant part of its cost (around 2 seconds) is due to the process of dispatching the DBMS-aglet from the client applet to the aglet router on the Web server and from there to the database server. In the case of the CORBA approach, the first query is slightly more expensive than the one in the JMA approach because of the overhead of initializing the necessary ORB classes and the binding to the application server. This overhead is quite significant (around 3.20 seconds) which can be clearly seen by comparing the response time of the initial and subsequent queries. Following the CORBA approach is the Java JDBC approach in which the response time of the initial query is increased by a considerable amount of time by the downloading of the JDBC driver from the Web server.



Graph 1: Performance of all approaches for initial and subsequent query (128 bytes result size)

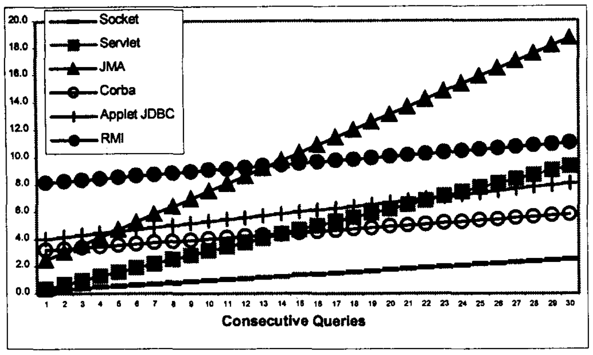
To our surprise, the RMI approach performs by far the worst for the initial query. We expected the RMI approach to exhibit better performance because, as opposed to the other RPC approaches, it does not involve the loading of any specific package during initialization time. The only way to explain this behavior is to attribute the increased response time to the interpreted method of RMI calls when binding the client applet to the application server.

For subsequent queries, the performance of the CORBA and RMI approaches dramatically improves, and becomes close to the best performance exhibited by the Socket approach. The reason is that the client applet is already bound to the remote application server and only a remote procedure call on the application server is required to query the database. For a similar reason, the Java JDBC applet approach also exhibits a significant performance improvement for subsequent queries - the JDBC driver is already downloaded and initialized at the client applet. Having the DBMS-aglet already connected to the remote database and ready to process a new query on behalf of the client applet, the JMA approach also improves its response time for subsequent queries. However, this response time is the worst from all the other approaches. We attribute this to two reasons. First, the two required messages to implement subsequent queries have to be routed through the aglet router, and second, a mobile agent is not a stand-alone process and it does not receive full CPU time.

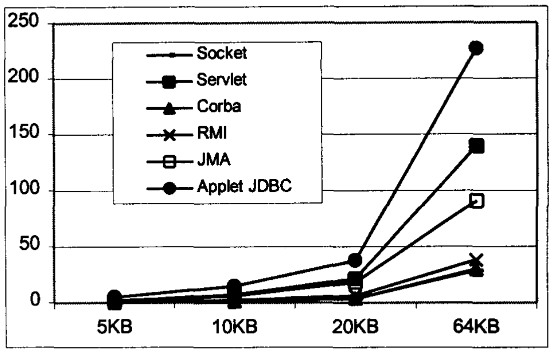
On the other hand, the Java Servlet approach improves only slightly its performance because the steps for executing a subsequent query do not differ from the ones for the initial query. The minor improvement is due to the fact that any subsequent URL connections from the client applet to the Web server require less time since the address of the Web has already been resolved in the initial query.

In order to better illustrate the *scalability* of each approach, we plotted in Graph 2 the average time required by each approach to query the database for a number of consecutive requests using the formula: For n consecutive queries, the average time required is the sum of (a) the average response time for one initial query, and (b) n-1 times the average response time for a subsequent query.

As shown in Graph 2, the socket approach is the most efficient for any number of consecutive queries. Despite its



Graph 2: Average performance for up to 30 consecutive queries (128 bytes of result size)



Graph 3: Subsequent Query

good performance for initial queries, the Servlet approach does not scale well since the response time for subsequent queries almost matches the response time for initial queries. Likewise, the JMA approach scales very badly given that its response time for subsequent queries is the worst of all the approaches. The CORBA, Java JDBC applet, and RMI approaches appear to scale well, however, the RMI approach appears less attractive due to its worst performance of all the approaches for initial queries.

Graph 3 illustrates the sensitivity of each approach to the size of query results. Due to space limitations, we show here only the results for subsequent queries. The results for initial queries are similar.

The first striking observation is that the response time of the Java JDBC applet and JMA approaches increases exponentially with query result sizes larger than 20KB. The Java JDBC applet approach performs by far the worst for increased result size. This can be explained by the fact that in JDBC rows from a query result are retrieved one at a time. Specifically, to retrieve one row from the query result, the client must call a method on a Java ResultSet object, which is mapped on the remote database server through the Gateway. Consequently, for a large size of query result, a large number of those remote calls have to take place. In that case, large query results not only increase dramatically the response time but they also increase the Internet traffic.

The bad scaling of the JMA approach can be explained in the same way as the bad performance of the Servlet approach. Both mobile agents and servlets do not execute as stand-alone processes, and therefore, they do not receive full CPU time and heavily depend on the supporting execution environment. The other RPC approaches exhibit acceptable performances (close to linear for sizes above 20KB) with the CORBA approach being slightly better. As indicated above, the implementation of RPC calls in CORBA is much faster compared to RMI's one.

VII. Conclusions

In this experimental paper, we have implemented, evaluated, and compared all currently available Java-based approaches for Web database connectivity. Our comparison was based on the performance of query processing, the transparency of communication and extensibility.

The results of our comparison showed that the CORBA approach is the most transparent to communication, extensible and easy to develop, while its performance is comparable to the best performing approach that employs sockets. Hence, it offers the best promise for the development of large Web applications.

In our study, we confirmed the desirable properties of the emerging mobile agents technology, that is, of high extensibility and transparency at a relatively low development effort. But, at the same time, our study provided an insight to potential scalability problems with the currently available mobile agent implementations. The JMA approach cannot support interactions that require movement or exchange of large amounts of data such as large number of consecutive queries with increased size of query result. Hence, it is necessary to develop more efficient mobile agent infrastructures, if the full potential of mobile agents is to be explored. As part of our future work, we investigate the possibility of merging mobile agents and the CORBA technology in order to facilitate a scalable and efficient Web database connectivity.

References

[1] E. Anuff. *Java Sourcebook.* Whiley Publishing, 1996.

[2] S. P. Hadjiefthymiades and D. I. Martakos. A Generic Framework for the Development of Structured Databases on the WWW. *Fifth Int'l WWW Conference*, May 1996.

[3] C. G. Harrison, D. M. Chessm, A. Kershenbaum. Mobile Agents: Are they a good idea? Research Report, IBM Research Division, 1994.

[4] G. Helmayer, G. Kappel, and S. Reich. Connecting Databases on the Web: A Taxonomy of Gateways. *Eighth Int'l DEXA Conference,* Sept. 1997.

[5] S. Papastavrou, P.K. Chrysanthis, G. Samaras, and E. Pitoura. An Evaluation of the Java-based Approaches for Web Database Connectivity. CSD Technical Report. University of Pittsburgh, Mar.2000.

[6] S. Papastavrou, G. Samaras, and E. Pitoura. Mobile Agents for WWW Distributed Database Access. Fourteenth *IEEE Int'l Conference on Data Engineering,* Feb. 1999.

[7] Sun Microsystems Inc., Java Development Kit, <http://java.sun.com/jdk>.

[8] Sun Microsystems Inc., *JDBC drivers,* <http://java.sun.com/products/jdbc/drivers.html>.

[9] Visibroker for Java: Programmer's Guide, Version 3.0. Borland, <http://www.visigenic.com>.