

# A Generic Database Web Service

Erdogan Dogdu

TOBB Economics and Technology University  
Computer Engineering Department  
Ankara, Turkey  
edogdu@etu.edu.tr

Yanchao Wang and Swetha Desetty

Georgia State University  
Department of Computer Science  
Atlanta, Georgia, USA

**Abstract**— Web applications built using Java technologies usually access relational databases via JDBC API. This requires a database system specific JDBC driver to be installed on the application side. On the other hand, a paradigm shift is taking place in web application architectures. Future web applications will be built around Service-Oriented Architectures (SOA) where applications will be assembled using remote “web services” components. These newly assembled applications will provide functionalities using remote web services components over Internet via XML messages. Same paradigm shift will eventually apply to the data communication between applications and databases. Future applications will utilize standard “database web services” for data storage and querying requirements. Here we propose a new architectural model and present a prototype “database web service” for relational database systems. In our model, web applications do not need to deal with database drivers but leave the driver-oriented communication to a database web service. Our database web service eliminates the need for installation, maintenance, and other issues involved in maintaining JDBC drivers in distributed web application development.

**Keywords:** Web services, SOAP, JDBC, distributed databases.

## I. INTRODUCTION

Extensible Markup Language (XML) is emerging as the new standard for data and message exchange over Internet [4]. New distributed application integration frameworks such as “Web Services” are based on XML standards [5]. Web Services framework allows applications to exchange data and provide remote services using XML formatted data and messages.

Web services framework consists of the following standards: (1) Extensible Markup Language (XML) for formatting data and messages, (2) Simple Object Access Protocol (SOAP) for exchanging messages, (3) Web Services Description Language (WSDL) for describing services, and (4) Universal Description, Discovery, Integration (UDDI) for providing service directories. All of these standards are based on XML [5].

Current web services framework only provides an infrastructure for building distributed applications utilizing

web services infrastructure. Many real-world business requirements such as integrity, automation, transactional support, and security issues are not completely dealt with yet in the current framework. Ongoing work is trying to address these issues [6].

Most of data in the web is currently held in relational database systems today. Web applications simply provide an interface to query and present that data in HTML format to the end user; web browsers interpret and display the HTML tagged data in an intended format. Therefore, the data presented in HTML format is not usable by other applications; it is more intended for directing the browser to present data for human reading. On the other hand, providing direct access to databases is not feasible due to security, performance, scalability, and reliability issues. There is a need for a standard mechanism to allow applications to access databases just like users access data on databases via web browsers.

Earlier, ODBC and JDBC provided a standard application programming interface (API) for applications to access relational database systems in a uniform way [1]. An application can access a remote or local relational database using these APIs as long as there is a vendor-specific driver provided for the intended database system. Currently, almost all database systems have ODBC or JDBC compliant drivers developed (there is also a JDBC-ODBC bridge to access ODBC-accessible databases via JDBC API) [2]. Problem with this approach is that a client application that wants to access a JDBC-accessible database needs to get a hold of the relevant driver for the database system and install on the client side. It is clear that this is not feasible for large-scale distribution of applications. If drivers get updated, this even complicates the issue with many problems: driver updates, maintenance of multiple versions, backward compatibility to name a few. Problem is exactly the same as in the case of early client-server based applications. Web provided a solution to this by a multi-tier application architecture where client is scaled down to a thin-client model with only presentation layer of the application on the client side, and the database access is only done in an application server on the server side.

Today's computing devices are getting smaller continuously. Wireless-connected personal digital assistants (PDAs) and Java-enabled cell phones are ubiquitous. These devices with their limited resources are not capable of handling complex database drivers. Applications developed for small devices like these need other mechanisms to access data seamlessly. Database web service provides an option; an application developed for these devices can access remote databases via web services, and this approach requires only a small XML messaging client.

In this paper we address the need for universal access to data resources without complicated installation and maintenance issues. Specifically, we utilize the newly arrived Web Services approach for application integration and interoperability for data access. To this end, we developed a generic database web service. This web service is deployed on a server where the database is installed. And, querying and updates on the database are enabled via the database web service. Client on the other hand only needs a web services client application that will only send and receive messages in XML format (SOAP messages).

Section 2 discusses the architecture of this new model where access to relational database is provided via database (JDBC) web service. Section 3 presents the current implementation, a JDBC Web Service, and its advantages are discussed. We discuss implementation and evaluation of JDBC-WS in section 4. Section 5 presents the related work and we conclude in section 5.

## II. ACCESSING RELATIONAL DATABASES VIA WEB SERVICES

Relational databases can be accessed by general-purpose Java applications using standard JDBC API [1]. This requires applications to use a JDBC driver specifically developed for JDBC access to the database system that needs to be accessed. Therefore, wide-distribution of such an application requires the driver to be also shipped to the client. Future changes to the driver require driver updates for every client.

Problem is even more complicated if the application needs to access many different relational database systems. Basically, this requires distribution and maintenance of many drivers in client-side.

Here we propose a new approach for applications to access database systems. It is based on Web Services framework. In this approach, client is reduced to a thin-client, just like web applications took the client-side computations to server side. In this case, database drivers are the problem, so they are placed on a server and moved away from the client. And, applications are provided a new interface to access database systems in a uniform way, similar to JDBC API provided a uniform access mechanism for all relational database systems. Figure 1 depicts the proposed architecture.

In this architecture, client accesses a "database web service" using the standard web services access mechanism, which is sending and receiving XML SOAP messages over standard Internet protocols to request database (web) services.



Fig. 1 Web Services-based access to relational databases

## III. JDBC WEB SERVICE

We implemented a database web service called JDBC Web Service (JDBC-WS). JDBC-WS will provide a standard JDBC-like interface that can be used by any Web Services client. JDBC-WS is accessed via a URL that is the connection string needed in JDBC. JDBC-WS clients do not need a database driver unlike usual JDBC clients. In the case of JDBC-WS, specific database and driver are completely transparent to the client. As long as the same web service interface (methods, parameters) are provided, database system and driver can be changed completely.

In the standard JDBC-based clients, a username and password needs to be provided for the database connection. Here, JDBC-WS can either authenticate the user once (and keep a session for consecutive operations) or every operation can be authorized via username/password passed along SOAP messages.

JDBC-WS accepts standard SQL queries embedded within SOAP messages. Following are some examples:

Scenario 1: To query the names of authors from the author table in a database, XML message to be sent to JDBC-WS is:

```

<query>
  SELECT name FROM author
</query>
  
```

And, response expected will be:

```

<result>
  <row>
    <name>James Gosling</name>
  </row>
  <row>
    <name>Jeff Ullman</name>
  </row>
  <row>
    <name>Donald Knuth</name>
  </row>
  ...
</result>
  
```

Scenario 2: To insert a new author name to the author table in a database, XML message is:

```

<query>
  INSERT INTO author (name)
  VALUES ("Jim Gray")
</query>

```

And, if the record is inserted to the table successfully, the response expected will be:

```

<result>>true</result>

```

Using a standard XML-interface to access relational and tabular databases, client applications access a number of different database systems using similar JDBC-WS interfaces and do not assume anything about specifics of the data source. Data sources are interchangeable and replaceable as long as the same interface is provided. This approach provides a complete distributed database framework for the client applications. Figure 2 provides a general overview of the distributed databases framework. JDBC-WS provides over multiple kinds of data sources (please note that JDBC 2.0 and above provide access to any tabular data resource such as spreadsheets and flat files).

JDBC-WS provides a number of advantages over standard JDBC access solutions:

1. No driver installation or distribution in client-side.
2. Client is resilient to driver changes, updates.
3. Client is resilient to database system changes. JDBC-WS hides all database system and corresponding driver installations from client.
4. If a "JDBC connection pooling" solution is provided in the server side, this is again completely hidden from client side and server side changes can be done anytime to increase the system performance as needed.

This web services approach can in effect be made transparent to the end user who can develop their applications using JDBC. They only need to make a few minor changes like the packages they include and the connection string. This can be achieved by providing client classes that override the existing JDBC classes. For example executing a query in the JDBC application would actually create an XML SOAP request message with the SQL query embedded in the message and that would be sent via HTTP to the remote server where it will be executed. The main advantage of this is that application programmers can use legacy JDBC-based applications with the new JDBC-WS implementation without making many changes.

The disadvantage of this approach could be that client and server side needs to process XML messages via standard XML parsing techniques. On the other hand, client side driver initiation, connection establishment is eliminated altogether. Therefore, performance evaluation of this approach needs to be done against standard JDBC-based access.

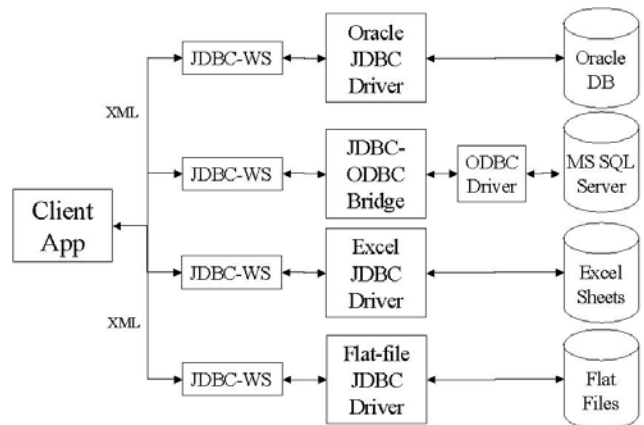


Fig. 2 Distributed data sources and databases via JDBC-WS

#### IV. IMPLEMENTATION AND EVALUATION

A prototype JDBC-WS package has been developed [9] using Java and Java Web Services Developer Pack from Sun Microsystems [3]. Package utilizes the external configuration files to set up the system for any data resource such as relational databases, spreadsheets, or flat files, without recompilation. In the configuration file, system admin specifies the data source driver, its location, connection parameters, connection pooling configuration, and other JDBC-WS setup parameters. Current implementation's architecture is depicted in Figure 3. Current implementation follows the JDBC API specification [1]. It implements the DriverManager, Connection, ResultSet, and Statement interfaces. These interfaces basically provide a gateway for accessing

Current implementation is using Oracle 10g database as the data source and the Oracle provided JDBC drivers are used in the testbed. JDBC-WS API we developed is used by the testbed to access the JDBC data sources to query the database.

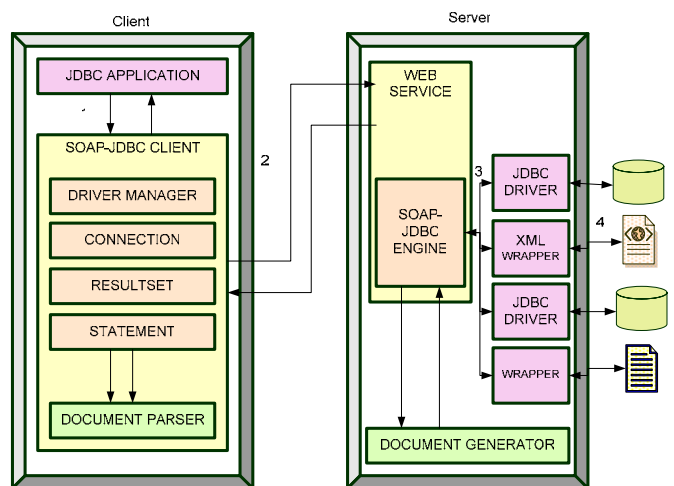


Fig. 3. SOAP-JDBC Architecture

System is currently being tested and evaluated against a standard JDBC implementation. A wide-variety of scenarios are going to be evaluated for both cases, such as queries with short results vs. long results, select queries vs. update queries, standalone vs. concurrent access, etc. System will be also be evaluated to report performance over different data resources.

Below we present some preliminary results (Figure 4). Figure shows the response times of queries for results of different sizes. The three lines in the figure are obtained by querying tables with different column numbers (5, 10, and 15 columns). In each case, tables are queried to obtain results with 50 to 500 rows increasingly. Since the results are tagged by JDBC-WS to convert into XML format, query result document becomes larger than the actual result size. This is apparent in the case of 15 column data source. As the number of rows in the result increases, response time of the query increases dramatically because of the number of tags that need to be generated and transmitted to the client.

## V. RELATED WORK

Providing uniform access to database resources is not a new issue. Distributed and federated databases have tried to address the issue earlier [7]. Current web services infrastructure however presents new opportunities in distributed application integration and interoperability of heterogeneous devices and platforms. Certainly, all software applications in future will in one-way or the other will deal with web services either to provide or to receive services to or from remote applications. Database systems are one of them due to the widespread use of databases anywhere.

Earlier, a Java specific solution, called RmiJdbc, is developed by ObjectWeb (open source middleware) to provide remote access to relational databases over Java RMI (Remote Method Invocation) technology [8]. It is very similar to our system in that database driver is hidden behind an RMI server and the client accesses the database via RmiJdbc server. Although similar to our work, RmiJdbc is very restrictive in the choice of programming language, only Java can be used to program both client and the server. On the other hand, JDBC-WS is programming language and platform independent. One can develop a JDBC-WS version for a Oracle database via C++ or Java, and the client can be developed in any language of choice as long as there is a supporting web services invocation client such Apache Axis, or .NET.

Table 1 summarizes the differences between RmiJdbc and JDBC-WS.

## VI. CONCLUSIONS AND FUTURE WORK

Web services framework has a promising future in distributed computing area due to the ubiquity of Internet

and overwhelming industry support for web services. Web services will bring the true interoperability and application integration in a language and platform agnostic manner. Databases have provided excellent data storage and querying mechanisms. A natural approach is to provide database access over web services.

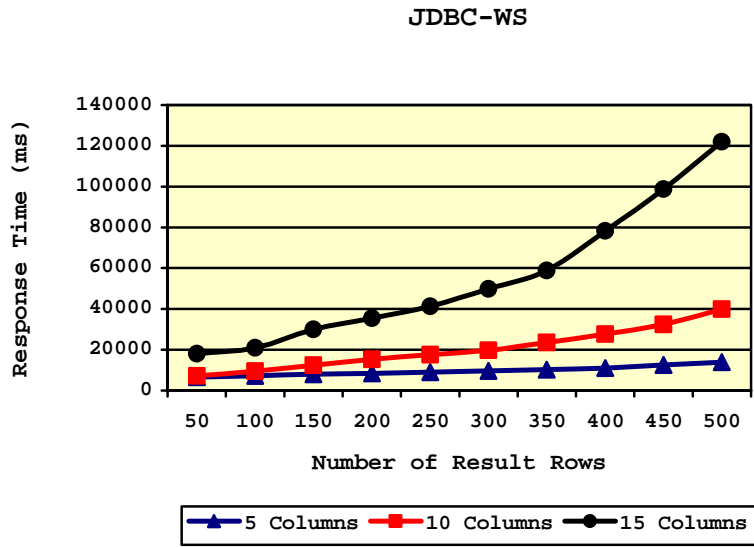
We have developed a new database access framework in that database servers are hidden behind a new database web service. Database web service provides a uniform SQL-based querying interface and query results are returned to the client in XML format. We have developed a prototype version of database web service called JDBC-WC. JDBC-WC, when installed for a database server, provides a database and driver transparent interface to the database for any client application that can implement SOAP messaging. Currently, the system is being evaluated and tested.

The work presented in this paper is part of a larger framework we are working on, where JDBC-WC will be used as a standard interface for any data source with SQL querying capabilities. In the extended framework we will address the following issues among others: transaction processing (web services composition), concurrency control, distributed query evaluation and optimization, database replication, distributed integrity constraint and constraint enforcement.

Acknowledgements: We would like to thank the Department of Computer Science at Georgia State University for providing support to conduct this research.

## REFERENCES

- [1] JDBC™ Data Access API, <http://java.sun.com/products/jdbc/>.
- [2] JDBC Drivers, <http://industry.java.sun.com/products/jdbc/drivers>.
- [3] Java Web Services Developer Pack, Sun Microsystems, <http://java.sun.com/webservices/>
- [4] Extensible Markup Language, <http://www.w3c.org/XML/>.
- [5] Web Services Activity, <http://www.w3c.org/2002/ws/>.
- [6] Web Services Transaction Specification (WS-Transaction) <http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/>.
- [7] Amit Sheth and James Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Surveys* 22(3): 183-236 (1990)
- [8] RmiJdbc, <http://rmijdbc.objectweb.org/index.html>.
- [9] Dessety, Swetha, "SOAP-JDBC: A Bridge Between Heterogeneous Clients and Data Sources", MS Thesis, Georgia State University, 2003.



**Fig. 4** Response times of JDBC-WS queries for different data sizes.

TABLE I. COMPARISON OF RMIJDBC AND JDBC-WS

	<b>RmiJdbc</b>	<b>JDBC-WS</b>
Database vendor independence	Yes	Yes
Database driver independence	Yes	Yes
Heterogeneous client and server platforms	No	Yes
Programming language independence	No	Yes
Support for Web Services Standards	No	Yes
Support for mobile devices	Yes	Yes