

Database Middleware and Web Services for Data Distribution and Integration in Distributed Heterogeneous Database Systems

Han-Chieh Wei

Computer Science Department
University of Central Arkansas
Conway AR, USA

Travis Godfrey

Computer Science Department
University of Central Arkansas
Conway AR, USA

Abstract - *As the database environment changing, database is no longer accessed only by using SQL to communicate with DBMS as in the two-tier client-server architecture. In modern enterprise distributed systems, applications are developed and deployed as components in n-tier architecture. Database system is integrated with related applications and can only be accessed via interfaces provided by the application server or middleware. Moreover, in most of the middleware systems such as CORBA, J2EE, and RPC, clients and servers are tightly-coupled and client applications can only invoke methods by using proprietary communication protocols. As a consequence, data integration is no longer achievable by applying existing database middleware systems. What we dealing now are not just different DBMS vendors, dialects of SQL, or variants of data models, but also heterogeneous interfaces and communication protocols defined in various distributed paradigms. In this paper, we propose ADIM, an autonomous data integration middleware, with WebService for data integration in heterogeneous environments.*

Keywords: Database Middleware, Data Integration, Distributed Databases.

1 Introduction

Enterprises have been very interested in the decentralization of processing while achieving an integration of the information resources within the geographically distributed systems of databases. A distributed database is a collection of multiple logically interrelated database distributed over a computer network. Compared with centralized databases, distributed databases have the advantages of increased reliability and availability, better performance, local autonomy, expandability, and sharability [12]. Nowadays, distributed databases in organizations commonly are heterogeneous, i.e., data are organized and managed by a mix of different data management systems from different vendors and different operating systems that use different network protocols. In essence, the data source in such

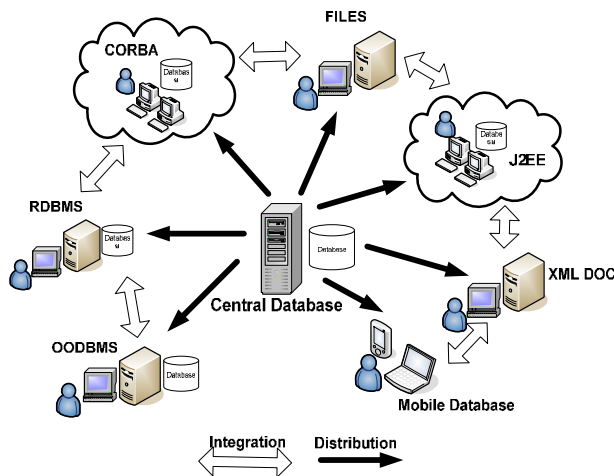
federated databases may consist of not only multi-vendor relational databases, but also non-relational databases such as object-oriented databases, ASCII data organized in flat files, or even XML databases. Such autonomous data sources have no ability to relate data from these heterogeneous data sources within the organization. The difficulties result from the diverge of data types, data representations, data manipulation languages (DML), data definition languages (DDL), transaction management, and security. These difficulties cause serious problems for data distribution and integration in enterprise applications.

Middleware is a layer of software whose purpose is to mask heterogeneity and to provide a convenient programming model to application programmers. Middleware is represented by processes or services in a set of computers that interact with each other to implement communication and resource sharing for distributed applications. With the same purpose, database middleware systems are used to integrate heterogeneous data sources distributed over computer networks. Client applications serviced by the database middleware are provided with uniform and transparent view and access interfaces to the distributed data. Whenever an application wants to access data managed in a heterogeneous distributed database, instead of writing code to establish connections to each local and remote data source and writing proprietary APIs for each data source, the client applications use local API's for data access provided by the database middleware. There is no need to modify or re-write the applications if there is replacement or insertion of new data sources.

Currently there are two main categories of database middleware systems: *database mediator* and *database gateway*. The database mediator establishes connections to data sources and use wrappers to access and translate the information from the data sources into a global data model. The database gateway establishes a point-to-point connection to one remote database and importing data into the local DBMS (Oracle, Informix, Sybase). Both solutions have been adopted in commercial database

systems to solve the heterogeneous distributed DBMS problem.

However, as the database environment changing, database is no longer accessed only by using SQL to communicate with DBMS as in the two-tier client-server architecture. In modern enterprise distributed systems, applications are developed and deployed in 3-tier or n-tier architecture [2,3]. Database system is integrated with related applications and can only be accessed via interfaces provided by the application server or middleware. Moreover, in most of the middleware systems such as CORBA, J2EE, and RPC, clients and servers are tightly-coupled and client applications can only invoke methods by using proprietary communication protocols. Figure 1 illustrates an example of distributed environment. As a consequence, data integration is no longer achievable by applying existing database middleware systems. What we dealing now are not just different DBMS vendors, dialect of SQL, or different data models, but also interfaces and communication protocols defined in various distributed paradigms.



In this paper, we present an autonomous database middleware approach for data integration and distribution in a distributed heterogeneous environment. In section 2, we start with investigating the requirements for such database middleware. The proposed architecture and implementation is presented in section 3. Section 4 gives a brief overview of the related work. We finally conclude the paper and give an outlook of the future work in section 5.

2 Background and Middleware Requirements

2.1 Types of distributed database systems

Distributed database systems (DDBS) can be characterized with respect to the degree of *local autonomy*, the degree of *heterogeneity*, and the degree of

distribution. For local autonomy, if there is no provision for the local site to function as a stand-alone DBMS, then the system has no local autonomy. On the other hand, the system is *fully autonomous* if the individual database systems are stand-alone DBMSs, which know neither the existence of other DBMSs nor how to communicate with them. What exists in between is the *semi-autonomous* system in which each DBMS operates independently and participates in a federation to share its local data. This type of distributed database systems is commonly categorized as *federated DBMS* [12] where there is a global view or schema of the federations of databases that is shared by applications and users at each site. For *heterogeneity*, it may occur in various forms in distributed database systems, ranging from operating systems, programming languages, and communication protocols to variations of data models, query languages, and transaction management protocols. If all participating data servers follow the same form, the DDBS is called *homogeneous*; otherwise it is called *heterogeneous*. For data distribution, the data is either stored at one site or physically distributed over multiple sites and communicate each other over the networks.

Our proposed middleware approach is targeting at the federated heterogeneous distributed database systems. The data is distributed over multiple sites. Based on the data model, data is managed by different data servers which can be a full-fledged database server, an application server, or even a file server providing access to flat files. These servers may provide different access interfaces and communicate with different protocols. Before presenting the proposed middleware architecture, we first investigate the requirements for a middleware to process data distribution and integration in a distributed heterogeneous database system.

2.2 Global database schema

An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers. A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent. For distributed database system, the *access transparency* hides differences in data representation and how a data source is accessed, where the *location transparency* hides where data source is located. To achieve data integration in a heterogeneous distributed database with access and location transparencies, the middleware layer imposes a *global data schema* on top of the individual schema used by each data source. The global schema stores not only the global database structures but also the information of data distribution such as address and specifications of vertical and horizontal fragmentations for each site. The global schema also needs to include the access interface

information for each data source. This information enables the applications to transparently access the data sources, no matter if they are stored locally or remotely, or can be accessed by means of SQL or APIs. With this mechanism, client applications serviced by the middleware system are provided with a uniform view and access interface to the data sets stored by each data source.

2.3 Client Interfaces

To access data in a distributed database system, the middleware has to provide interfaces for user client or applications. The possible interfaces mostly include GUI for users to pose queries against the database schema and visualize their results, and APIs for programmer to develop complicated applications. The query is represented in a generic format and evaluated in the middleware. The middleware is responsible to connect to the data source, get the result, and convert the data into the format that can be displayed to the client.

2.4 Data Server Interfaces

The middleware has to implement interfaces for the data sources to be integrated. These interfaces must overcome the heterogeneous communication protocols as well as heterogeneous platforms including operating systems and programming languages. Since the results are typically returned in different formats, the interfaces should translate them into the reference data model which is used inside the middleware. In summary, the middleware has to provide a homogeneous view across heterogeneous and distributed application interfaces.

2.5 Middleware Query Processing

For data integration, client applications issue queries against the global database schema. Providing with location and access transparencies, client applications have no idea where the data located, how to connect to the data server, or what are the access interfaces. All these tasks are handled by the middleware. The database middleware decomposes the query and find corresponding sites for each data item which can handle the request. The middleware makes query execution plans and performs join operations if necessary.

2.6 Security

There are four levels of security can be defined [10] in the middleware architecture: the login right of the application to the middleware, the access rights of the middleware to the global schema, the login right of the application to the foreign data server, and the privileges of the application to the foreign database.

2.7 Scalability

The scalability is evaluated as the ability to add heterogeneous data sources. The middleware should be able to handle the addition of data source and data servers to the system and update the global catalog without having to modify the client applications.

3 Overall Architecture

To solve the problems described in section 1 and fulfill the requirement listed in section 2, we propose the middleware architecture Autonomous Data Integration Middleware (ADIM) as depicted in Figure 2. There are four main components in ADIM: *Global database Schema, Client, Server, and Web Service*.

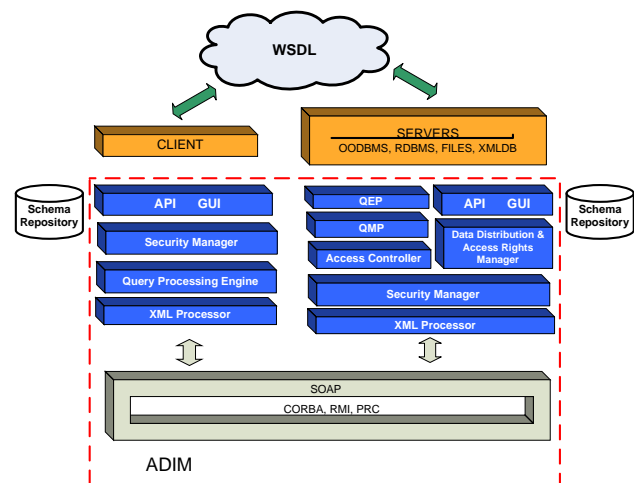


Figure 2 Autonomous Data Integration Middleware

3.1 Global Database Schema

As mentioned in previous sections, each distributed data source may implement different data models. To deal with this heterogeneity, ADIM uses XML document to represent the global database schema. Because of its extensibility, the XML document can also include the distribution information such as rules for horizontal and vertical fragmentations, addressees, and access interfaces and protocols of distributed data servers. Figure 3 shows part of the global schema represented by XML document. In Figure 3, the XML document describes the table definition, distributed sites and the local server and access services.

3.2 Client Components

There are four logical layers in the client component: API/GUI, Security Manager, Query Processing Engine (QPE), and XML processor. For user clients, after logging into the middleware, the GUI displays the global database schema which is tailored from the access rights of the local middleware to the

distributed database systems. The user can use the GUI to issue queries against the global schema and visualize the results. The queries will be parsed in QPE. Provided with the distribution information from the global schema, requests will be sent to data sources at which the requested data located. As we will discuss in the following section, all these requests are sent by using SOAP [13] messages which are prepared in XML processor. While receiving the results, the XML processor parses the SOAP message for QPE to process and the results either displayed on the GUI or returned to the application program.

3.3 Web Services

As mentioned earlier, in a distributed heterogeneous database system, different data sites may provide different access interface for same request and communicated with different protocols. For the database gateway approach, it provides a point-to-point connection to one type of database. In this case, each data source needs to implement one gateway for each of the other data sources and communicate with their proprietary protocols. As we can see from Figure 4(a), while this approach solves the heterogeneity problem, it introduces exponential complexity and limits the system scalability. In ADIM, all the server access interfaces are wrapped with Web Services and accessed with SOAP, as shown in 4(b), which makes the system scalable in linear complexity. Web services [15] describes a standardized way of integrating internet based applications using the XML, SOAP [13], WSDL [16] and UDDI [14] open standards over the Internet. Web services allow different applications from different sources to communicate with each other without time-consuming custom coding, and because all communication is in XML, Web services are not tied to any one operating system, programming language, or communication protocols. In this way, it does not matter whether the data is accessed through a database server, Java server, CORBA server, RPC server, or even a file server, ADIM can communicate uniformly with SOAP. Figure 5 gives an example of the SOAP messages for query and response. Currently there are seven web services defined for the data server: *UpdateSchema*, *Execute*, *ExecuteQuery*, *ExecuteUpdate*, *PrepCommit*, *Commit*, and *Abort*. Because of the space limit, we will not describe the detail implementation here. The service names are pretty much self-explained.

```

<Tables>
<Table id="Student">
<fields>
<field>
<name>name</name>
<type>string</type>
<length>2</length>
</field>
<field>
<name>address</name>
<type>string</type>
<length>50</length>
</field>
<field>
<name>city</name>
<type>string</type>
<length>10</length>
</field>
<field>
<name>state</name>
<type>string</type>
<length>2</length>
</field>
</fields>
<Sites>
<Site>
<CorbaHostName>192.168.1.101</CorbaHostName>
<CorbaPort>1981</CorbaPort>
<Fragmentation_type>Horizontal</Fragmentation_type>
<Fragmentation_Rule>city in ('conway', 'little rock', 'north
little rock', 'sherwood')</Fragmentation_Rule>
<servicetype>Corba</servicetype>
</services>
<service>Query</service>
<service>insert</service>
<service>load_table</service>
</services>
</Site>
</Sites>

```

Figure 3 Database Global Schema

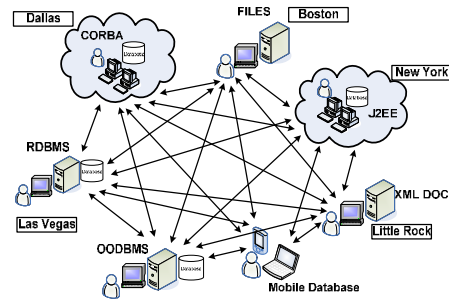


Figure 4 (a) Database Gateway Approach, Exponential API

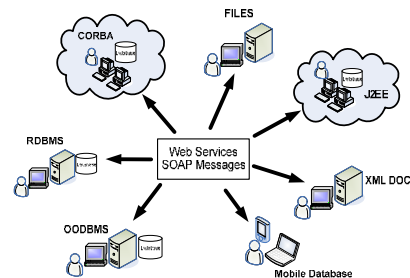


Figure 4 (b) Web Service Wrapper, Linear Growth

The server can develop more local services and update the WSDL and also the global schema to be available for the middleware. We implement a *SOAP Wrapper Generator* to automatically generate Java programs as a bridge to the backend services. As depicted in Figure 6(a), having CORBA as the backend service, the generator reads input from the IDL file and a configure file containing the host address and port information, it will then generate Java programs acting as SOAP service call to the CORBA and the also the WSDL for middleware to access the service. The calling process is shown in Figure 6(b). Figure 7 depicts the Java code

generated from the CORBA IDL and also the WSDL definition.

```

Soap message
Request
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:select
xmlns:ns1="urn:dos_soap_db_WSDL">
<param1 xsi:type="xsd:string">"select name from
student"</param1>
</ns1:doubleAnInteger>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:selectResponse
xmlns:ns1="urn:dos_soap_db_WSDL"
SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:int">"1"</fields>"n"<field
id=name>travis</field>"n</fields>"</return>
</ns1:doubleAnIntegerResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Figure 5 Sample SOAP message

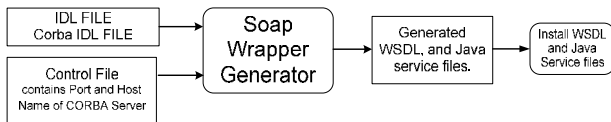


Figure 6 (a) SOAP Wrapper Generator

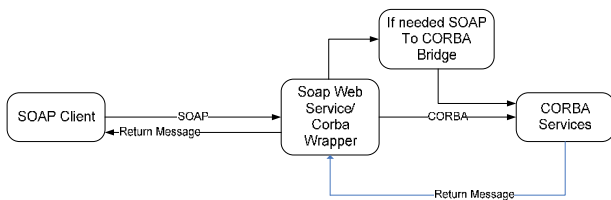


Figure 6 (b) Call to the CORBA Wrapper

3.4 Server Component

There are also four logical layers in the server component: Query Execution Processor (QEP), Query Mapping Processor (QMP), Security Manager, and XML Processor. The XML processor parses the SOAP message received from the client. The security manager send login information to the server on behalf of the client for authentication and authorization check. The Query Mapping Processor (QMP) maps the query against the local definition for the Query Execution Processor (QEP) to extract data from the data source. The results will be translated into the global schema in QMP and prepared in XML processor as SOAP response message and send back to the client. The GUI module provide an interfaces for administrator at central control site to define data distribution specifications, such as sites

addresses, horizontal and/or vertical fragmentation rules, and site access authorization. These specifications will then be added to the global schema, sent to each site, and stored in the middleware schema repository.

```

module DBApp
{
interface DB
{
boolean createTable( in string tblInfo );
boolean loadTable( in string tblData );
boolean insert( in string tblRow );
string select( in string tblSelection );
};

};

public class dos_soap_dbService {
public org.omg.CORBA.Object myobj;

public dos_soap_db_Service()
{
String file = "C:\myfile.ctd";
String port = "8081";
String host = "localhost"
ORB_CONNECT orbConnect = new ORB_CONNECT(String file, String port, String host);
this.myobj = orbConnect.obj;
}

public boolean createTable (String tblInfo) throws Exception
{
return myobj.createTable(tblInfo);
}

public boolean loadTable (String tblData) throws Exception
{
return myobj.loadTable(tblData);
}

public boolean insert (String tblRow) throws Exception
{
return myobj.insert(tblRow);
}

public int delete (String tblConditional) throws Exception
{
return myobj.delete(tblConditional);
}
}

```

Figure 7 Web Service Wrapper

4 Related Work

There are related proposed architectures for heterogeneous data integration [1,5,6,7,8,9]. Oracle distributed database system uses the database gateway approach (transparent gateway) to access non-Oracle database systems. MOCHA [10] and [8] are middleware based integration architectures that implement distributed query processing for data and function integration in federated database systems. CoDIMS-G [5] offers a standard based communication paradigm based on Web Service and distributed query processing for distributed multi-database system in the grid environment. SkyQuery [9] is a Web Service based integration architecture for the integration of astronomy data archives available on the internet. It adopts Web Services as the main communication and encapsulation interface both for the query processor and data sources wrappers.

5 Conclusion and Future Work

In modern enterprise systems, databases and applications are distributed over the network for the benefits of better performance, economy, reliability, and availability. Nowadays, distributed databases in organizations commonly are heterogeneous, i.e., data are organized and managed by a mix of different data model and data management systems from different vendors and

different operating systems that use different network protocols. With the emergence of component-based engineering [6], enterprise systems may consist of application systems built with different distributed paradigms, such as EJB, .NET, or CORBA. Databases encapsulated in such application systems can only be accessed via provided APIs and connected using proprietary protocols. In such highly heterogeneous environment, data integration by using traditional federated database systems or database gateway are no longer feasible or sufficient.

In this paper, we propose a database middleware architecture ADIM (Autonomous Data Integration Middleware) with WebService for data distribution and integration in distributed heterogeneous database systems. The data schema integration is achieved by XML based global schema which is used by QPE and QMP in the client and server components to translate and exchange metadata between data sources. The heterogeneity of access interfaces and protocols provided by each distributed paradigm is resolved by adopting WebService as the communication model for uniformity of communication protocol in HTTP, message exchange in SOAP for service request and response, and service definition and description in WSDL. We also develop the SOAP wrapper generator which can generate Java code to wrap the server API into Web Services. We have implemented a prototype of ADIM in the distributed environment described in section 2. We are now continuing working on the improvement of query optimization and XML document join. We are also working on the two-phase-commit service and distributed scheduler to extend ADIM to allow database update operations.

Acknowledgements

This research was partially supported by the University of Central Arkansas URC grant.

6 References

- [1] Rakesh Agrawal, Dmitri Asonov, and R. Srikant., Enabling Sovereign Information Sharing Using Web Services, in *Proceedings of the 2004 ACM SIGMOD international conference on Management of Data*, June 2004.
- [2] Francisco Álvarez Cavazos and Juan Carlos Lavariega Jarquín , A 3-Tiered Client-Server Distributed Database System Component-Based, in *Proceedings of the winter international symposium on Information and communication technologies*, 2004
- [3] A. Davis, J. Parikh, and W.E. Weihl, EdgeComputing: Extending Enterprise Applications to

the Edge of the Internet, in *Proceedings of the 13th ACM international World Wide Web conference*, May 2004.

- [4] Fernando de Ferreira Rezende and Klaudia Hergula, "The Heterogeneity Problem and Middleware Technology: Experiments with and Performance of Database Gateways," in *Proc. VLDB Conference*, pp.146-157, 1998.
- [5] V. Fontes *et al.* "CoDIMS-G: a Data and Program Integration Service for the Grid," in *Proceedings of the 2nd Workshop on Middleware for Grid Computing*, ACM 2004.
- [6] Ondrej Galik and Tomas Bures, "Connecting middlewares: Generating Connectors for Heterogeneous Deployment", in *Proceedings of the 5th International Workshop on Software Engineering and Middleware*, ACM, 2005.
- [7] D. Heimbigner and D. McLeod, "A Federated Architecture for Information Management." *ACM Trans. Office Information System*, Vol. 3 No. 3, pp. 253-278, July, 1985.
- [8] Klaudia Hergula, DaimlerChrysler AG, and Theo Harder, "A Middleware Approach for Combining Heterogeneous Data Sources - Integration of Generic Query and Predefined Function Access," in *Proceedings of the 1st International Conference on Web Information Systems Engineering*, 2000.
- [9] Tanu Malik, Alex S. Szalay, Tamas Budavari, and Ani R. Thakar. "SkyQuery: A Web Service Approach to Federate Databases," *Proceedings of CIDR 2003*.
- [10] M.R. Martinez, N. Roussopoulos. "MOCHA: A Self-Extensible Database Middleware System for Distributed Data Sources," *ACM SIGMOD Record* , *Proceedings of the 2000 ACM SIGMOD international conference on Management of data SIGMOD 2000*, Volume 29 Issue 2.
- [11] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, F.D. Ngoc, Exchanging Intensional XML Data, in *Proceedings of the 2003 ACM SIGMOD international conference on Management of Data*, June 2003.
- [12] M. T. Özsu, P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.
- [13] SOAP, XML Protocol. <http://www.w3.org/2000/xml/Group/>.
- [14] UDDI Technical Committee Specification. http://uddi.org/pubs/uddi_v3.htm, 2003.
- [15] Web Services. <http://www.w3.org/2002/ws>.
- [16] Web Services Description Languages, WSDL. <http://www.w3.org/TR/wsdl>.