An ODBC CORBA-Based Data Mediation Service

Paul L. Bergstein
Dept. of Computer and Information Science
University of Massachusetts Dartmouth, Dartmouth MA
pbergstein@umassd.edu

Keywords: Data mediation, data integration, ODBC

Abstract

A serious problem facing many organizations today is the need to use information from multiple data sources that have been developed separately. Conflicts in the structure and semantics of these disparate data sources create major obstacles to effective use. We have previously described our data mediation approach to this problem and our implementation of an RMI-based mediation service with JDBC drivers. Here we report our implementation of a CORBA-based mediation service with ODBC drivers, making mediation convenient for a wider class of applications. An application can switch to any data source registered with the mediator simply by plugging in our drivers and specifying the mediation service as the data source. No other code needs to be rewritten.

1. Introduction

A serious problem facing many organizations today is the need to share information among systems that have been developed separately. The information sharing may be within the organization or with external partners. In either case, the heterogeneity of the data creates major obstacles to effective sharing of information. Conflicts may exist in both the structure and the semantics of the data involved. Furthermore, the structure and semantics of a data source may change over time.

Historically, there have been a variety of approaches to this problem [1-4]. The simplest approach is to build a messaging system for each pair of data sources that wish to exchange data. The messaging system translates data to and from the agreed message format at each end. However this approach doesn't scale well if there are many systems that want to participate in the sharing, since a messaging system is needed for each pair. There is another problem as well. The metadata documenting the structure and semantics of an enterprise's data that is required to build the messaging system is a very valuable resource, but it may get lost in the translating code.

Another approach is to define standards. The standardization approach takes several forms. For example, we could standardize the data sources, making the data homogeneous. While seemingly simple, this approach has proven impossible in practice. Since different data sources are designed to be used in different environments, they are heterogeneous for good reasons, and nobody can agree on a common standard.

Standardizing the message format is another possibility. This approach is not new, but has recently been receiving widespread attention in the form of defining standard DTD's for exchanging data in XML format [5]. Given the level of effort in this direction, we expect to see quite a bit of success, especially within limited and well defined domains. On the other hand, prior attempts to define standard message formats have generally failed due to lack of agreement on the format's structure and semantics. Note that agreement to use XML does not solve this problem. It is still necessary to agree on the structure (what tags to use) and the semantics (what the tags mean). Also, two systems exchanging information through a standard message format may lose information and/or precision during the exchange that could have been preserved using a custom format.

The data mediation approach relies on a common ontology that can be used to describe the structure and semantics of each of the systems that wish to participate in the information sharing. A data mediator uses these descriptions to perform any necessary translation between systems exchanging information. In a variation of this approach, a shared view is created, and the mediator translates queries written against the shared view. Mediation has the advantages that there is no need to agree on standard formats, the metadata is made explicit (so it may be reused), and translations only occur where the structure or semantics between two systems differ. In many situations, we believe that mediation will prove to be a better approach than standardization.

Our ODBC mediation service uses a layered architecture as shown in Figure 1. In the next section we will summarize the lower layers and core workings of our data mediator. Then we will describe the CORBA service and ODBC drivers which are built on top of the mediation functionality. We have previously reported on the RMI service and JDBC drivers[6]. The architecture for applications using the mediator is shown in Figure 2.

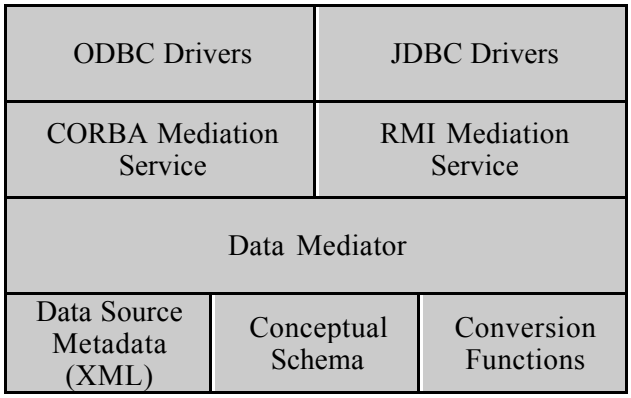


Figure 1

2. Background

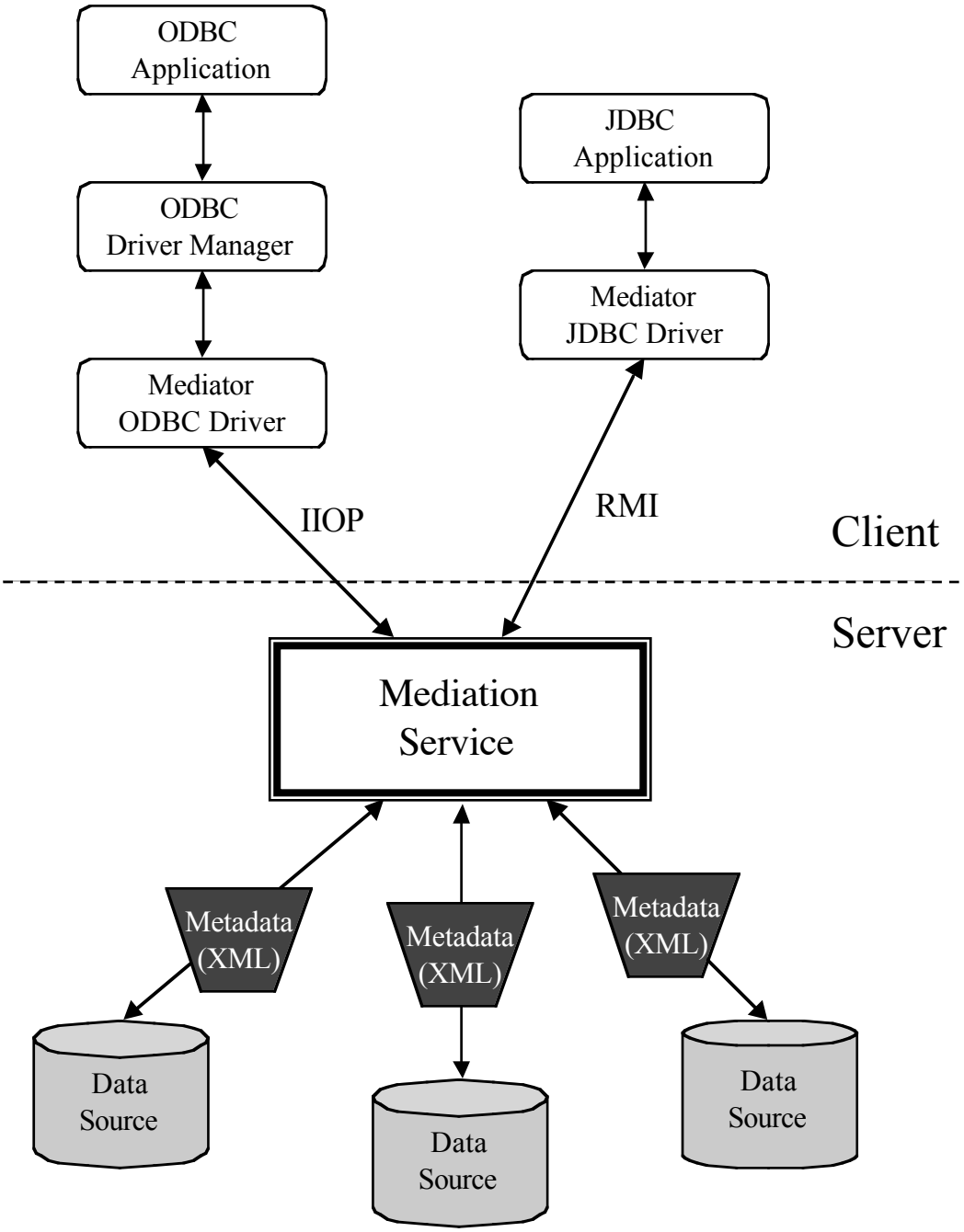
Our data mediator was originally based on the following scenario: Suppose a user who knows the schema of only their local database, System A, wishes to retrieve information from a foreign database, System B. They write a query against the schema of System A, but indicate that they would like to use System B as the data source. The mediator translates the query against System A into one or more queries against System B, executes the queries, and translates the results into the local format of System A.

In our current work we consider a slightly different scenario: Suppose an application using ODBC (or JDBC [6]), has been written to use a particular data source, System A, but we now want to use a different source, System B, with a different structure or semantics. We accomplish the change simply by plugging in our Mediator ODBC (or JDBC) driver in place of the System A driver. The application can now use the new data source without rewriting any code. Notice that plugging in the System B driver will only work if Systems A and B have identical structure and semantics, otherwise mediation is required.

2.1 Conceptual Schema

In our implementation, the common ontology is expressed as a shared conceptual schema, which includes both ordinary classes (e.g. University, Student) and domain classes (e.g. Money, Date). The attributes of ordinary classes have domain classes as their types. For example, Student might have an attribute called graduation-date with type Date. For each domain class, we specify subclasses (sub-domains) for the known representations. When a new data source is registered with the mediator, it will typically be necessary to add sub-domains for data representations that are unique to that data source.

The conceptual schema is never populated in our system. It is used only as a reference for defining the structure and semantics of the actual data sources. In particular, the conceptual schema is *not* used as an intermediate data representation when transferring data from one source to another. Instead, the mediator synthesizes a plan for direct conversion between the data sources based on their structures and semantics as defined by their individual mappings to the conceptual schema.


ODBC
Application
ODBC
Driver Manager
Mediator
ODBC Driver
JDBC
Application
Mediator
JDBC Driver
IIOP
RMI
Client
Server
Mediation
Service
Metadata
(XML)
Metadata
(XML)
Metadata
(XML)
Data
Source
Data
Source
Data
Source

2.2 Conversion Functions

The mediator uses a repository of functions for converting between representations within a domain. In our (java) implementation all conversion functions have the same interface. They take a java Properties object as parameter, and return a Properties object as the result, so they naturally support many-to-many mappings, e.g. (latitude, longitude) to (point, bearing, range). The repository is implemented as a java class with methods for each conversion function. The repository uses java introspection to search for suitable conversion functions.

2.2 Metadata

In order to register a data source with the mediator, a description of the data source (its metadata) must be supplied in XML format. For each data source there is a separate XML file prepared by someone familiar with that data source. Currently, the XML files are prepared manually, but we plan to develop tools to help generate these files. The metadata includes information required to connect to the data source as well as mappings to the conceptual schema that define the structure and semantics of the data source. We use XML[1] to map data elements of real databases onto attributes of ordinary classes in the conceptual schema. Each mapping to an attribute of an ordinary class includes the subdomain of the data element.

In the simplest case each data element of System A corresponds one-to-one with an element of the conceptual schema, which in turn corresponds one-to-one with an element of System B. If the conceptual schema contains an ordinary class called Employee with a salary attribute of type Salary, and System A has a Worker relation with a pay-rate attribute, then the XML file for System A would map Worker/pay-rate to Employee/salary and it would also map pay-rate to one of the subdomains of Salary such as Annual/USDollars or Monthly/Euros[2]. Similar mappings from System B provide the mediator with the information needed for translation.

Mappings between the conceptual schema and an actual database are not always one-to-one. Suppose that in the conceptual schema Professor's have a phone-number attribute of type PhoneNumber, but in the actual database Instructor's have area-code, exchange, and extension attributes. For the mediator to work, the PhoneNumber domain class must have a subdomain, say ACEE, for the area-code/exchange/extension representation of phone numbers, with attributes corresponding to the three parts of a phone number. Each of the area-code, exchange, and extension attributes is mapped to the Professor/phone-number attribute (and also to the appropriate attribute of the ACEE subdomain). The mapping (from actual to conceptual) is many-to-one.

If another database uses the same representation of phone numbers, so we have mappings like:

A: (code, exchg, ext) → phone-number
B: (area, exg, extension) → phone-number

then the translation will not use conversion functions (even if the data elements have different names). In other cases, such as:

A: (latitude, longitude) → position
B: (point, bearing, range) → position

a conversion function is required.

Sometimes data source isn't a very good match for the conceptual schema. This is likely to happen, for example, when a new data source is added after the conceptual schema has been completed. Consider, for example, a conceptual schema that has entity classes for full-time students and part-time students, and a data source with graduate students and undergraduate students. In this case we map attributes, e.g. gpa, from both graduate and undergraduate students to attributes of both full-time and part-time students. Additionally, we supply conditions that determine, for example, which graduate students are part-time and which are full-time. These conditional mappings [5] are

[1] For brevity, in this paper we mostly describe mappings without showing the XML syntax since the XML is trivial but verbose.

[2] In theory, the issues of currency units and frequency of payment should be separate, but we combine them for the sake of simplicity in our implementation, in order to focus on more interesting concerns.

specified in both directions (to and from the conceptual schema).

2.3 Data Mediator

The data mediator manages the conversion function repository, the conceptual schema, and the data source metadata. It is responsible for synthesizing query and translation plans. When a query against the schema of System A is executed using System B as the data source, the mediator translates the query against System A into one or more queries against System B, executes the queries, and translates the results into the local format of System A. The details of our algorithm are beyond the scope of this paper, but will be reported elsewhere.

The mediator is implemented entirely in Java and uses JDBC to access the desired data source. Therefore, the mediator can be used to exchange data between any data sources that have JDBC drivers available, including most relational databases, all ODBC data sources (via a JDBC/ODBC bridge driver), and XML data (using an available XML JDBC driver).

3. CORBA Mediation Service and ODBC Drivers

The CORBA mediation service is a thin wrapper around the data mediator. The interface to the service consists of a single function that takes an SQL query, the schema name against which the query is written, and the name of the data source to be used to retrieve the data. It uses the mediator to produce a result set in the format of the specified schema. The service can utilize any JDBC data source (including any ODBC data source via the JDBC/ODBC bridge driver) that has been registered with the mediator.

Our ODBC drivers are implemented in C on linux for unixODBC and are designed for easy portability to other platforms. The use of CORBA allows us to easily access the mediator functionality implemented in Java from the ODBC drivers written in C. So far, our implementation is incomplete but we have implemented enough to demonstrate proof of concept. Specifically, we have working implementations of the SQLAllocHandle, SQLConnect, SQLSetStmtAttr, SQLBindCol, SQLExecDirect, SQLFetch, SQLFreeHandle, and SQLDisconnect driver API functions.

Under unixODBC, the drivers are configured with entries in two configuration files. In odbcinst.ini, an ODBC driver is mapped to its library, e.g:

```
[MEDIATOR]
driver = /usr/lib/libmediator.so
```

An entry in the system or user's odbc.ini file maps a data source name (DSN) to a driver, and supplies values for whatever other attributes are required by the driver. In the case of the mediator driver, the server, port, schema name, and data source name are required, e.g:

```
[A2B]
driver = MEDIATOR
server = sebago.cis.umassd.edu
port = 1956
schema = sysA
datasource = sysB
```

The *schema* and *datasource* are names of data sources that have been registered with the mediator. In principle, a data source could be registered with the mediator under any arbitrary name, but in order to avoid name collisions we use names similar to:

edu.umassd.cis.db1.

Queries and result sets are in the form of the specified *schema*, while the specified *datasource* is used as the actual data source.

When requesting a connection to the mediator, the username and validation strings that are supplied to SQLConnect must be a valid username/password pair for the *datasource* specified in the configuration file. We don't see any need for password protection of the mediator service itself, though we could easily add such protection if the need arises. As it stands, clients cannot access information through the mediator that they would not otherwise have access to, although the mediator will make the information accessible through a familiar interface and supply it in a much more convenient format.

Our SQLExecDirect implementation uses the mediation service to execute queries and produce result sets in the format of the specified schema.

Using our ODBC drivers, changes in data source are almost completely transparent to client applications. If we want to modify an application designed to use a particular data source so that it uses a different data source, it is only necessary to plug in our drivers and add the appropriate entries to the ODBC configuration. No other C/C++ code or SQL queries need to be rewritten. Using these drivers it also becomes trivial for application developers to allow users to select a data source of their choice at runtime (assuming the data sources have been registered with the data mediator).

4. Conclusions

There are numerous other researchers [1-4, 7-12] who have investigated mediation as a way of resolving structural and semantic conflicts between data sources. However, as far as we can determine, there are no previous reports of an ODBC based mediation service as described here.

The addition of the mediation service and ODBC driver layers to our architecture significantly enhances the utility of our mediation technology. The principal benefits are the ability to easily change the data source used by an ODBC application, and the ability to easily offer the user a choice of data sources at runtime.

We have tested our implementation on a small number of sample applications and found it to be highly effective. However, we are still working on improvements in several areas. We are working actively to complete a fully ODBC compliant set of drivers. We are also working to improve the underlying mediation technology.

Currently our mediator is able to successfully mediate most simple SQL queries that don't involve joins or sub-queries. We are currently working on enhancing the mediator so that it can handle queries involving joins. We are particularly interested in cases where relations have been decomposed differently in the local and foreign databases, so that the joins required in the foreign database are different than those specified for the local database. We are also working to improve the handling of where clauses in the mediator.

5. References

[1] E. Sciore, M. Siegel, and A. Rosenthal, "Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems", *ACM Transactions on Database Systems*, vol. 19(2), June 1994, pp. 254-290.

[2] G. Wiederhold, "Mediators in the Architecture of Future Information Systems", Readings in Agents, Eds. M. N. Huhns and M. P. Singh, San Francisco, CA, USA: Morgan Kaufmann, 1997, pp. 185-196.

[3] P. B. Lowry, "XML data mediation and collaboration: A proposed comprehensive architecture and query requirements for using XML to mediate heterogeneous data sources and targets," *34th Annual Hawai'i International Conference On System Sciences (HICSS)*, Mauii, Hawaii, January 3-6, 2001, pp. 2535-2543.

[4] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel, "Context interchange: new features and formalisms for the intelligent integration of information", *ACM Transactions on Information Systems*, vol. 17(3), July 1999, pp. 270.

[5] P. Bergstein and V. Shah, "Conditional Mapping in Data Mediation", *Proceedings of the International Conference on Information and Knowledge Engineering (IKE 2004)*. June 21-24, 2004, Las Vegas, Nevada, USA. CSREA Press 2004, ISBN 1-932415-27-0.

[6] P. Bergstein and A. Sikder, "A JDBC Data Mediation Service", *Proceedings of the International Conference on Information and Knowledge Engineering (IKE 2005),* pages 45-50, June 20-23, 2005, Las Vegas, Nevada. CSREA Press, ISBN 1-932415-81-5.

[7] L. S. Seligman and A. Rosenthal, "XML's Impact on Databases and Data Sharing", IEEE Computer, vol. 34(6), 2001, pp. 59-67.

[8] G. Neugebauer, "GLUE – Using Heterogeneous Sources of Information in a Logic Programming System", *Proceedings of the KI'97 Workshop on Intelligent Information Integration*, Freiburg, 1997.

[9] L. Serafini and F. Giunchiglia and F. Mylopoulos and P. Bernstein, "The Local Relational Model: A Logical Formalization of Database Coordination", *Proceedings of CONTEX'03*, 2003.

[10] H. Wache and H. Stuckenschmidt, “Practical Context Transformation for Information System Interoperability”, *Lecture Notes in Computer Science*, vol. 2116, 2001, p. 367.

[11] B. Ludäscher, A. Gupta, and M. Martone, “Model-Based Mediation with Domain Maps”, *17th International Conference on Data Engineering (ICDE ’01)*, Washington-Brussels-Tokyo, April 2001.

[12] C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu, “XML-based Information Mediation with MIX”, Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data: SIGMOD '99, Philadelphia, PA, June 1-3, 1999, SIGMOD Record, vol. 28(2), 1999, pp. 597-599.