

A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases

Young-Kwang Nam¹, Joseph Goguen², and Guilian Wang

¹ Dept. of Computer Science, Yonsei University, Wonjoo, Republic of Korea
yknam@dragon.yonsei.ac.kr,

² Dept. of Computer Science and Engineering, UCSD, La Jolla, CA 92093
goguen, guilian@cs.ucsd.edu

Abstract. This paper describes a metadata interchange approach for semi-automated integration of heterogeneous distributed databases. Our system prototype uses distributed metadata to generate a GUI tool for a meta-user (who does the metadata integration) to describe mappings between master and local databases by assigning index numbers and specifying conversion function names; the system uses Quilt as its XML query language. A DDXMI (for Distributed Database XML Metadata Interface) file is generated based on the mappings, and is used to translate queries over the virtual master database into sub-queries to local databases. An experiment testing feasibility is reported in which 3 different bibliography databases are integrated.

1 Introduction

It is often required to integrate and analyze data from multiple sources, e.g., in ecology, sociobiology, medicine, and electronic commerce. As stated in [23, 25], increasing standardization or adoption of ad hoc standards, such as Dublin Core [5], as well as metadata standards in domains such as bibliography [4], space, astronomy, geography, environmental science [13], and ecology [24], have achieved system, syntactic, structural, and limited semantic interoperability. Unfortunately, it is unrealistic to expect that integration can be done entirely through standardization. The major difficulty is that the data at different sources tends to be formatted in changing and incompatible ways, and even worse, represented under changing, incompatible and often implicit assumptions. For example, the bibliographical databases of different publishers may use different units of prices, different formats for author and editor names (e.g., full name or separated first name and last name), and the publisher name may be only implicit. Moreover, some data values in one schema may correspond to database or schema labels in another. Even worse, the same word may have a different meaning, and the same meaning may have different names. This implies that syntactical data and meta-data can not provide enough semantics for all potential integration purposes. As a result, the data integration process is often very labor-intensive and demands more computer expertise than most application users have. Therefore, semi-automated schemes seem the most promising, where mediation engineers

are given an easy tool to describe mappings between the global (global and master are used interchangeably in this paper) schema and local schemas, to produce a uniform view over the local databases.

Our approach, called DDXMI (for Distributed Database XML Metadata Interface), builds on that of XMI [28]. The master DDXMI file includes database or XML document name or location information, table column or XML path information, semantic information about table column or XML elements and attributes. A system prototype has been built that generates a tool for meta-users to do the meta-data integration, producing a master DDXMI file, which is then used to generate queries to local databases from master queries, and to integrate the results. This tool parses local DTDs, generates a path for each element, and produces a convenient GUI. The mappings assign indices to match local elements to corresponding master elements and to names of conversion functions. These functions can be built-in or user-defined in Quilt [9], which is our XML query language. The DDXMI is then generated based on the mappings by collecting over index numbers. User queries are processed by Quilt according to the generated DDXMI, by generating an executable query for each relevant local database.

This system is simple, since some of the most complex issues are handed off to Quilt, and it is easy to use, due to its simple GUI. The system is also flexible: users can get any virtual integrated database they want from the same set of data sources, and different users can have different virtual databases supporting their own applications.

2 Related Work

Many diverse solutions to data integration have been developed, although most of them are based on a common mediator architecture [27]. Mainly, they can be classified into structural approaches and semantic approaches. In structural approaches, the mediation engineer's knowledge of the application specific requirements and local data sources are assumed as a crucial but implicit input. The integration is obtained through a virtual global schema that characterizes the underlying data sources. On the other hand, semantic approaches assume that enough domain knowledge for integration is contained in the exported conceptual models, or "ontologies" of each local database. This requires a common ontology among the data source providers, and it assumes that everything of importance is explicitly described in the ontologies; however, these assumptions are often violated in practice.

Tsimmis [26], MedMaker [22] and MIX [2] are structural approaches. A common data model is used, e.g., OEM (Object Exchange Model) in Tsimmis and MedMaker, and XML in MIX. A view definition language is provided for the mediation engineer to define an integrated view that specifies how local data sources are integrated to the system, e.g., MSDL in Tsimmis and MedMaker and XMAS in MIX. MSDL and XMAS also act as query languages. All of these take a global-as-view approach. According to the integrated view definition, at

query time, the mediator resolves the user query into sub-queries to suitable wrappers that translate between the local languages, models and concepts, and global concepts, and then integrates the information returned from the wrappers.

In some other systems with structural approaches, users are given a language or graphical interface to specify only the mappings between the global schema and local schemas. The system will then generate the view definition based on these mappings. In Information Manifold (IM) [19, 26], description logic CARIN is used to specify local database contents and capabilities. IM has a mediator that is independent of applications, since queries over the global schema are rewritten to sub-queries over the local databases (defined as views over the global schema) using the same algorithm for different combinations of queries and sources. The most important advantage of local-as-view approaches is that it allows an integrated system built this way easily handles dynamic environments. Clio [14, 20] introduced an interactive schema-mapping paradigm in which users are released from the manual definition of integrated views in a different way from IM. A graphical user interface allows users to specify value correspondences, that is, how the value of an attribute in the target schema is computed from values of the attributes in the source schema. Based on the schema mapping, the view definition is computed using traditional DBMS optimization techniques. In addition, Clio has a mechanism allowing users to verify correctness of the generated view definition by checking example results. However, Clio transforms data from a single legacy source to a new schema; it remains a challenge to employ this paradigm for virtual data integration of multiple distributed data sources. Xyleme [8] provides a mechanism for view definitions through path-to-path mappings in its query language, assuming XML data.

Recently, in order to realize semantic interoperability in the sense of allowing users to integrate data and query the system at a conceptual level, many efforts are being made to develop semantic approaches, including RDF (Resource Description Framework) [2], Knowledge Sharing Effort [16], Intelligent Integration of Information [15], the Digital Library Initiative [10], and Knowledge-based Integration [12, 17]. Several ontology languages have been developed for data and knowledge representation, and reasoning formalism to help data integration from semantic perspective, such as F-Logic [12, 17, 18], Ontolingua [11], XOL [4], OIL and DLR [7, 6]. But despite some optimistic projections to the contrary, the representation of meaning in anything like the sense that humans use that term, is far beyond current technology. The meaning of a document often involves a deep understanding of its social context, including how it is used, its role in organizational politics, its relation to other documents, its relation to other organizations, and much more, depending on the particular situation. Moreover, these contexts may be changing at a rather rapid rate, as may the documents themselves. These complexities mean it is unrealistic to expect any single semantics written in a special ontology language to adequately reflect the meaning of documents for every purpose. Ontology mediation approaches can be frustrating to users, due to the difficulty of discovering, communicating, formalizing and updating all the necessary contextual information

3 System architecture

The overall architecture of the DDXMI distributed database system is shown in Figure 1. We assume that all databases are in XML, either directly or through wrapping. The basic idea is that a query to the integrated system, called a master query, is automatically rewritten to sub-queries, called local queries, which fit each local database format using the information stored in DDXMI by the query generator. The DDXMI contains the path information and functions to be applied to each local database, along with identification information such as author, date, comments, etc. The paths in a master query are parsed by the query generator and replaced by corresponding paths of each local document, by consulting the DDXMI if there are paths for the master query. If not, a null query is generated for the corresponding path in the local query, which means that this query cannot be applied to that local database. Each local query generated will be sent to its corresponding local database engine, which will process the query and return the result for the master query. Of course, there may be duplicated answers, and/or the results of some of local queries may need to be joined. Such issues will be handled by the database engine in a future prototype.

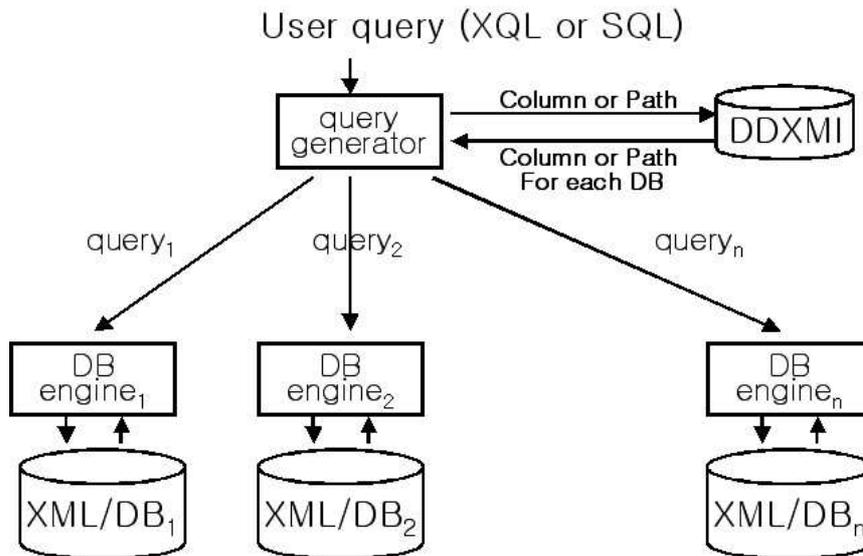


Fig. 1. System architecture

4 Distributed Database XML Interface (DDXMI)

4.1 DDXMI DTD

The DDXMI is an XML document, containing meta-information about relationships of paths among databases, and function names for handling semantic and structural discrepancies. The DTD for DDXMI documents is shown in Figure 2. Elements in the master database DTD are called source elements, while cor-

```

<!ELEMENT DDXMIA (DDXMI.header, DDXMI.isequivalent, documentspec)>
<!ELEMENT DDXMI.header (documentation,version,date,authorization)>
<!ELEMENT documentation (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT authorization (#PCDATA)>
<!ELEMENT DDXMI.isequivalent (source,destination)*>
<!ELEMENT source (#PCDATA)>
<!ELEMENT destination (#PCDATA)>
<!ELEMENT documentspec (document, (elementname, shortdescription,
                           longdescription, operation)*)*>
<!ELEMENT document (#PCDATA)>
<!ELEMENT elementname (#PCDATA)>
<!ELEMENT shortdescription (#PCDATA)>
<!ELEMENT longdescription (#PCDATA)>
<!ELEMENT operation (#PCDATA)>

```

Fig. 2. DDXMI DTD

responding elements in local database DTDs are called destination elements. When the query generator finds a source element name in a master query, if its corresponding destination element is not null, then paths in the query are replaced by paths to the destination elements to get a local query. (We will see that these may be more than one destination element.) For example, consider there are several book databases at different sites. The 'author' field in the master database may be represented as 'author', 'author-name', 'name', 'auth' element, etc. in different local databases. Then in the DDXMI, the 'author' source element matches with the destination element 'author', 'author-name', 'name', or 'auth' appropriate in each local database. More complex cases are discussed below.

4.2 How to generate a DDXMI

Since each database is in XML format, each document has its own DTD file. We assume that elements in local DTDs do not contain attributes. This implies that DTDs can be represented as n-ary trees. Our approach involves mapping paths in the master DTD to (sets of) paths in the local DTDs, though we often

speak of nodes instead of the paths that lead to these nodes. We match a node in the master DTD with nodes in local database DTDs, through numbering each node in the master DTD tree and then assigning these numbers to the node(s) with the same meaning in the local DTD trees. Hence nodes with the same number are involved with the same meaning. By collecting all nodes with the same numbers, the source and destination paths can be generated automatically, and the DDXMI can be easily constructed. An especially convenient special case is where an element in the master DTD matches one in a local database DTD, in that its field has the same meaning as the one in the master DTD. Elements in local databases should not appear in the DDXMI file if their meaning does not relate to any element in the DDXMI. The possible number of elements in the master DTD is the union of those in all the local database DTDs.

```

<!ELEMENT bib (book* )>
<!ELEMENT book (title, (author+ | editor+ ), publisher, price )>
<!ATTLIST book year CDATA #REQUIRED >
<!ELEMENT author (last, first )>
<!ELEMENT editor (last, first, affiliation )>
<!ELEMENT title (#PCDATA )>
<!ELEMENT last (#PCDATA )>
<!ELEMENT first (#PCDATA )>
<!ELEMENT affiliation (#PCDATA )>
<!ELEMENT publisher (#PCDATA )>
<!ELEMENT price (#PCDATA )>

```

Fig. 3. Book1.DTD file

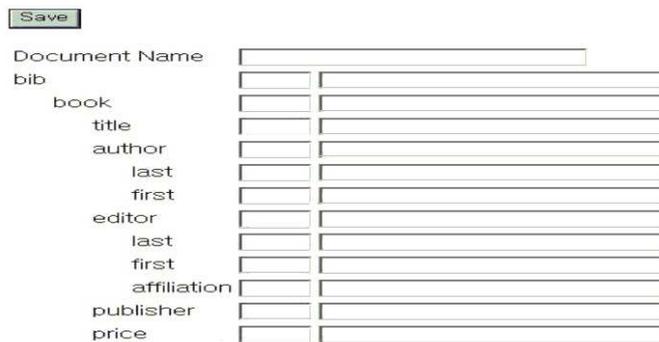


Fig. 4. Tree for Book1.DTD

For example, Figure 4 is generated from the parsed form of the Book1.DTD in Figure 3. The first column of Figure 4 is for entering indices for database DTDs. Then by collecting all nodes with the same index, the DDXMI source and destination elements are generated. Nodes without an index are not included in the master database. The document name is for the name of the local data document. The second column is for names of functions to resolve semantic issues.

Because local databases may have different structures for the same element, we have to provide some mechanism to handle such cases. For example, a local document may represent author's names as full names, while the master document separates the first and last names. In that case, the answer from the local document must be separated if a query is to retrieve the first name of the author. We classify such cases according to the mapping cardinality in the following subsections.

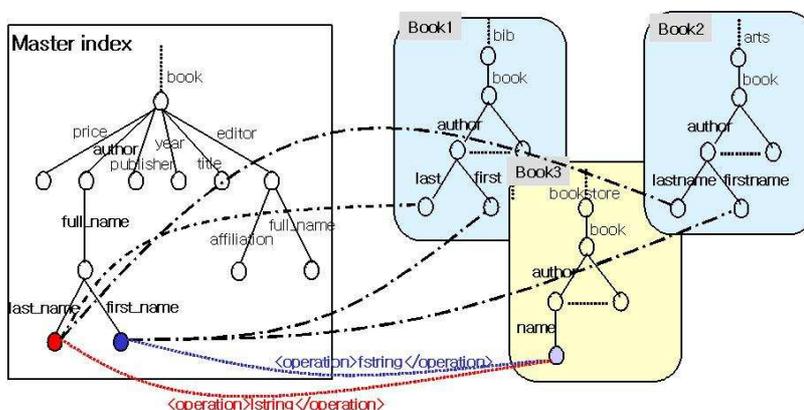


Fig. 5. N to one mapping example

N to one mapping If two or more nodes of the master DTD correspond to one node in a local database, then the node in the local DTD will have more than one index numbers. For example, the `first_name` and `last_name` nodes in the master DTD tree in Figure 5 are mapped to the `full name` node in the Book3.DTD tree. In this figure, only the Book3.DTD has full names; the others use separated first name and last name. The separation function names, `fstring` and `lstring`, are included in the DDXMI file for the full name node of the Book3.DTD. A portion of DDXMI for handling this mapping is shown in Figure 6.

One to N mapping Another case is where one node in the master DTD is mapped to several nodes in a local document. For example, the `editor` name in

```

<source>/book/author/full_name/first_name</source>
  <destination>/bookstore/book/author/name</destination>
<source>/book/author/full_name/last_name</source>
  <destination>/bookstore/book/author/name</destination>
<documentspec>
  <document>book.xml</document>
  <elementname>/book/author/full_name/first_name</elementname>
    <operation>lstring</operation>
  <elementname>/book/author/full_name/last_name</elementname>
    <operation>fstring</operation>

```

Fig. 6. A portion of DDXMI for N to one mapping case

the master DTD may be represented separately in a local document, Book1, as in Figure 7. Here the function `con` is used to concatenate the first and the last element to get the full name.

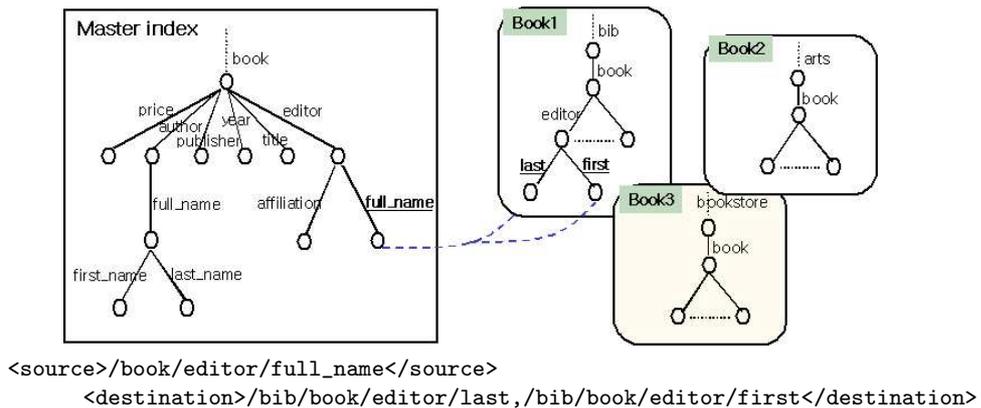


Fig. 7. One to N mapping case example

One to One with semantic functions As mentioned earlier, conflicts may be caused by using different reference systems. For example, the price field in Figure 4 may use the dollar currency, but the Book3.DTD in Figure 8 may use Canadian currency or be represented in cents. Some mechanism is required to translate such representations. For the price element, when the query is parsed, it is replaced by the function `price/100` in order to get the dollar unit answer.

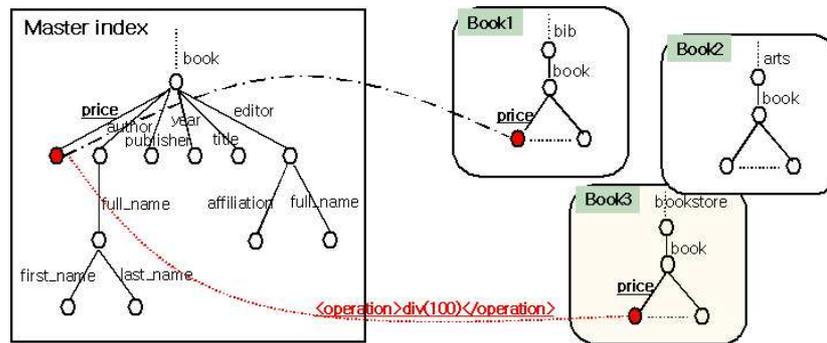


Fig. 8. An example for one to one mapping with a function

4.3 Replacing paths in a query

In XML query languages, there are two kinds of paths, relative and absolute. A path name after '/' means an absolute path, and one after '/../' means a relative path. Every master element has a corresponding path name with local element names if it has the same index number. But some elements in the master DTD may have a longer path name in the local query. When assigning the index number, some nodes in the middle of the tree may be skipped without assigning a number. Then the node in that path will include the skipped node name in the path. In Figure 9, the price node in the master DTD and in a local DTD have the same node name, but the index number is not assigned in the price_info node in the local DTD, so that the price element in the query is replaced by the "/price_info/price" path.

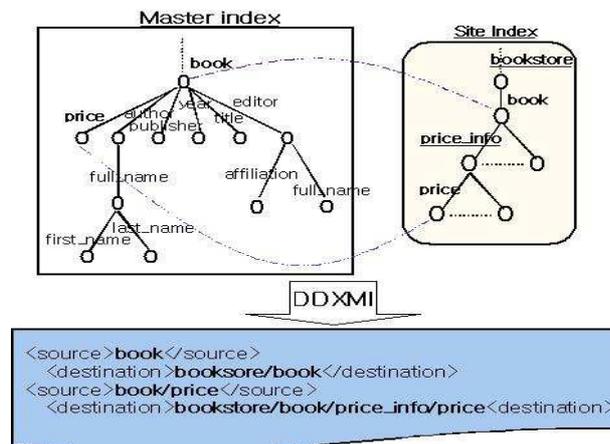


Fig. 9. An example for getting path name

5 DDXMI Generation example

Assume we are going to build a library master database consisting of DTD paths as shown in Figure 10. Figure 11 shows three indexed DTD trees. The same

```

0    book.xml
1    /book
11   /book/price
12   /book/author
121  /book/author/full_name
1211 /book/author/full_name/first_name
1212 /book/author/full_name/last_name
13   /book/title
14   /book/year
15   /book/publisher
16   /book/editor
161  /book/editor/affiliation
162  /book/editor/full_name

```

Fig. 10. A Master Library database DTD paths with the index numbers

[Book1.Index]		[Book2.Index]		[Book3.Index]	
0	Book1.xml	0	Book2.xml	0	Book3.xml
1	/bib/book	1	/arts/book	1	/bookstore/book
11	../price	12	../author	11	../price div(100)
12	../author	1211	../author/firstname	12	
1211	../author/first	1212	../author/lastname	1211	../name fstring
1212	../author/last	13	../title	1212	../name lstring
13	../title	15	../publisher	13	../title
15	../publisher				
16	../editor				
161	../editor/affiliation				
162	../editor/last				
162	../editor/first				

Fig. 11. The indexed DTD tree for Book1.DTD, Book2.DTD, and Book3.DTD tree.

index number is given to elements that have the same meaning, and each index sequence is different. Some nodes in the DTD tree have no number, since the master index does not include it. Book1.DTD and Book3.DTD have a 'price' node but Book2 DTD doesn't. If some query includes the 'price' element, then

no query would be generated for Book2.XML since Book3.DTD doesn't have a price element. Based on the Figure 10 index assignment, the DDXMI file will be generated automatically by collecting paths with the same numbers. Using this file, users can modify or update the DDXMI file easily, or the file can be re-generated by reassigning the index numbers.

6 Query generation and execution examples

Quilt is the XML query language of our prototype implementation. For the above three XML documents, three query generation and execution examples are given. Kweelt, developed in University of Washington, is used as a query engine to execute quilt queries. In our system, when a user enters a query name and pushes the enter button, then the local queries are generated. Figure 12 and 13 show examples of Quilt queries with their generated local queries and their executions for 2 mapping cases.

```

FOR $a IN document("book1.xml") //book//author
RETURN <author > $a/last, $a/first</author>
EXEC

FOR $a IN document("book2.xml") //book//author
RETURN <author > $a/lastname, $a/firstname</author>
EXEC

{ split("-", $str)[1]}
FUNCTION lstring($str)
{ split("-", $str)[2]}
FOR $a IN document("book3.xml") //book//author
RETURN <author> fstring($a/name), lstring($a/name)</author>
EXEC

<?xml version="1.0"?>
<author> Benjamin Franklin</author>
<author> Herman Melville</author>
<author> Plato Joon</author>
<!-- end of document -->

```

Fig. 12. The query generation and execution in case of one to N mapping

7 Conclusion and remaining issues

We have proposed a system for resolving both structural and semantic conflicts in a distributed database system using a GUI tool for generating a file called



Fig. 13. The query generation and execution in case of N to one mapping

DDXMI that contains information about data semantics. DTD trees are generated automatically, allowing a metadata engineer to assign the same index number to nodes with the same meaning and to name any necessary semantic functions for resolving representational differences. Then the DDXMI file is generated by collecting the index numbers. A master query from an end-user is then translated to queries to local databases by looking up corresponding paths in the DDXMI. Finally the results from local databases are integrated, using the named semantic functions.

Our DDXMI system has been implemented for the NT operating system under Java servlet server and JavaCC compiler. Any databases or documents represented in XML can be handled without rebuilding or changing the databases. The only requirement for users to employ this approach to integration is to be familiar with the XML data model.

However some issues remain to be investigated. First, if there are attributes in some DTD elements, they might correspond to just elements in other databases. This makes it difficult to generate queries for local databases. Helping users to handle redundancy will also be considered in our next prototype. In addition, we plan to migrate to XML schemas, instead of DTDs.

References

1. C. Baru, A. Gupta, B. Ludascher, R. Marciano, Y. Papakonstantinou, P. Velikhov, V. Chu. XML-Based Information Mediation with MIX. Exhibition program, ACM

- Conf. on Management of Data, SIGMOD'99, Philadelphia, 1999.
2. D. Brickley, R. Guha, eds. Resource Description Framework (RDF) Schema Specification. W3C Proposed Recommendation. March 1999. <http://www.w3.org/1999/TR/PR-rdf-schema>.
 3. K. Beard, T. Smith. A framework for meta-information in digital libraries. In Sheth A, Klas W (eds) Multimedia Data Management: Using Metadata to Integrate and Apply Digital Media. McGraw Hill: 341-365. 1998.
 4. V. K. Chaudhri, A. Farquhar, et al. OKBC: Open Knowledge Base Connectivity 2.0. Technical report KSL-98-06, Knowledge System Laboratory, Stanford, July 1997.
 5. Online Computer Library Center, Inc 1997 Dublin Core Metadata Element Set: Reference Description. 1997. Office of Research and Special Projects, Dublin, Ohio. http://www.oclc.org:5046/research/dublin_core/
 6. D. Calvanese, G. Giacomo, and M. Lenzerini. Ontology of Integration and Integration of Ontologies. Proc. of the 2001 Description Logic Workshop (DL 2001). 2001.
 7. D. Calvanese, G. Giacomo, M. Lenzerini, D. Nardi, R. Rosati. Description Logic Framework for Information Integration. In Proc. of Principles of Knowledge Representation and Reasoning (KR'98), pages 2-13, 1998.
 8. S. Cluet, P. Veltri and D. Vodislav. Views in a Large Scale XML Repository. 27th International Conference on Very Large Data Bases (VLDB 2001), Roma, Italy, 2001.
 9. D. Chamberlin, J. Robie and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. Proceedings of WebDB 2000 Conference, in Lecture Notes in Computer Science, Springer-Verlag, 2000.
 10. Digital Library Initiative. http://www.cise.nsf.gov/iis/dli_home.html.
 11. A. Farquhar, R. Fikes and J. Rice. The Ontiliqua Server: A Tool for Collaborative Ontology Construction. International Journal of Human-Computer Studies, 46:707-728, 1997.
 12. A. Gupta, B. Ludascher, M. E. Martone. Knowledge-Based Integration of Neuroscience Data Sources, 12th Intl. Conference on Scientific and Statistical Database Management (SSDBM), Berlin, Germany, IEEE Computer Society, July, 2000.
 13. O. Gunther, A. Voisard. Metadata in geographic and environmental data management. In Sheth A, Klas W (eds) Multimedia Data Management: Using Metadata to Integrate and Apply Digital Media. McGraw Hill: 57-87. 1998.
 14. L. M. Haas, R. J. Miller, B. Niswonger, M. Tork Roth, P. M. Schwarz and E. L. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. Bulletin of IEEE Computer Society Technical Committee on Data Engineering. 1999.
 15. Intelligent Integration of Information. <http://mole.dc.isx.com/I3>.
 16. Knowledge Sharing Effort. <http://www-ksl.stanford.edu/knowledge-sharing>.
 17. B. Ludascher, A. Gupta, M. E. Martone. Model-Based Mediation with Domain Maps. 17th Intl. Conference on Data Engineering (ICDE), Heidelberg, Germany, IEEE Computer Society, April 2001.
 18. B. Ludascher, R. Himmeroder, G. Lausen, W. May, and C. Schleppehorst. Managing Semistructured Data with FLORID: A Deductive Object-Oriented Perspective. Information Systems, 23(8):589-613, 1998.
 19. A. Y. Levy, A. Rajaraman, J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In Proc. of VLDB. Pp 251-262, Bombay, India, 1996.
 20. R. J. Miller, L. M. Haas and M. A. Hernandez. Schema Mapping as Query Discovery. Proceedings of the 26th VLDB Conference. Cairo Egypt, 2000.
 21. R. J. Miller. Using Schematically Heterogeneous Structures. SIGMOD '98 Seattle WA, USA. ACM 0-89791-995-5. 1998.

22. Y. Papakonstantinou, H. Garcia-Molina, J. Ullman. MedMaker: A Mediation System Based on Declarative Specifications. *Data Engineering (ICDE)*. 1996.
23. C. Parent and S. Spaccapietra. Issues and Approaches of Database Integration. *Communications of the ACM*, 41(5):166-178, 1998.
24. O. Reichman, J. Brunt, J. Helly, M. Jones, M. Willig. A Knowledge Network for Biocomplexity: Building and Evaluating a Metadata-based Framework for Integrating Heterogeneous Scientific Data. <http://www.nceas.ucsb.edu/>
25. A. P. Sheth. Changing focus on interoperability in information systems: from system, syntax, structure to semantics. In M F Goodchild, M J Egenhofer, R Fegeas and C A Kottman (eds), *Interoperating Geographic Information Systems*. Kluwer. 1998.
26. J. D. Ullman. Information Integration Using Logical Views 6th Int. Conference on Database Theory, LNCS 1186, 1997.
27. G. Wiederhold. Mediators in the Architecture of Future Information System. *IEEE Computer* 25:3, pp. 38-49, 1992.
28. XML Metadata Interchange (XMI). <http://www-4.ibm.com/software/ad/library/standards/xmi.html>.