A Metadata Tool for Retrieval from Heterogeneous Distributed XML Documents

Young-Kwang Nam[1], Joseph Goguen[2], and Guilian Wang[2]

[1] Dept. 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. A metadata approach for retrieval from heterogeneous distributed XML documents is given. A prototype system uses distributed metadata to generate a GUI data integration tool, for describing mappings between master and local DTDs, by assigning index numbers and specifying conversion functions. A DDXMI (Distributed Document XML Metadata Interchange) file is generated based on the mappings, and used to translate queries over the virtual master document into sub-queries to local documents. Quilt is the XML query language. An experiment testing feasibility is reported using three bibliography documents. The system runs under NT, using Java servelets and JavaCC.

1 Introduction

As more and more information sources become available online, it is often required to retrieve information from multiple sources, e.g., in ecology, sociobiology, medicine, and e-commerce. Hence convenient access to multiple, heterogeneous information sources through an integration mechanism is very desirable. Another trend is towards semistructured data formats such as XML, to combine the advantages of structured and unstructured data by imposing some structure on free text. XML is an easily processed format for extracting information, without the rigidity of a relational database [7].

Our DDXMI approach (for Distributed Documents XML Metadata Interchange), builds on XMI [19]. The master DDXMI file includes the XML document name or location, XML path information, and semantic information about XML elements and attributes. A prototype system has been built that generates a tool for meta-users to do meta-data integration, producing a master DDXMI file, which is then used to generate queries to local documents 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, which link local elements to corresponding master elements and to the names of conversion functions. These functions can be built-in or user-defined in Quilt [6], 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 document. With DDXMI, users can get all the answers within the master DTD scope by just writing a query for the virtual master DTD instead of for each document.

This system is relatively simple, since some of the most complex issues, such as distributed queries and query optimization, 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 information system based on the same data sources, and different users can have different virtual information systems for their own applications.

2 Related Work

Besides data warehousing, many solutions to heterogeneity of multiple distributed data sources have been developed, based on the mediator architecture [18]. In structural approaches, the mediation engineer's knowledge of application specific requirements and local data sources are a crucial but implicit input. Integration is obtained through a virtual integrated 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 data source. This requires a common ontology among the data source providers, and assumes that everything of importance is explicitly described in the ontologies; however, these assumptions are often violated in practice.

ROBIN [11, 15], Tsimmis [17], MedMaker [16], MIX [1], and IIT Mediator [7] are structural, mostly global-as-view approaches. According to the integrated view definition, at query time the mediator resolves the user query into subqueries to suitable wrappers that translate between the local languages, models and concepts, and the global concepts, and then integrates the information returned from the wrappers. In some other structural approaches, users are given a language or graphical interface to specify only the mappings between the global schema and local schemas, from which the system generates the view definition. In Information Manifold (IM) [12, 17], the description logic CARIN is used to specify local document contents and capabilities. The IM mediator is independent of applications, since queries over the global schema are rewritten to sub-queries over the local documents (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 an integrated system built this way easily handles dynamic environments.

Clio [10, 13] 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 [5] provides a mechanism for view definitions through path-to-path mappings in the form of a set of pairs (abstract path, concrete path) in its query language, assuming XML data. Our prototype differs from Xyleme in its query language independence, and in using a local-as-view mapping description, which is translated to global-as-view when generating the corresponding DDXMI file. Hence it combines the virtues of both global-as-view and local-as-view approaches. The present paper differs from our previous paper [14] in its focus on problems relevant to information retrieval and documents, and in describing new features of the implemented system to handle mappings that involve attributes, whereas previously we could only handle mappings among elements.

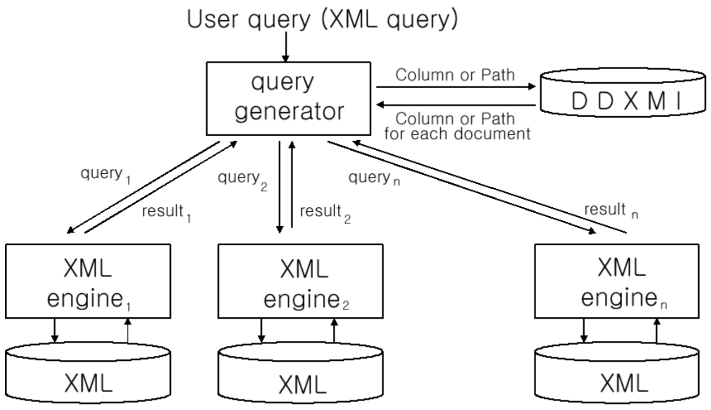


Fig. 1. DDXMI system structure for distributed documents

3 The Distributed Documents XML Interface (DDXMI)

An overview of the DDXMI system structure for retrieval from distributed documents is shown in Figure 1. We assume all documents are in XML, either directly or through wrapping. The basic idea is that a query to the distributed documents, called a master query, is automatically rewritten to sub-queries, called local queries, which fit each local document format using the information stored in the DDXMI by the query generator. The DDXMI contains the path information and functions to be applied to each local document, along with identification information such as author, date, comments, etc. The paths in a master query are parsed by the query generator and corresponding paths of each local document are substituted, if there are paths for the master query, by consulting the DDXMI. If not, a null query is generated for the corresponding path in the local query, meaning that this query cannot be applied to that document. The result collects all the answers from each document.

3.1 The Structure of DDXMI

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

```
<!ELEMENT DDXMI (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. DTD for DDXMI

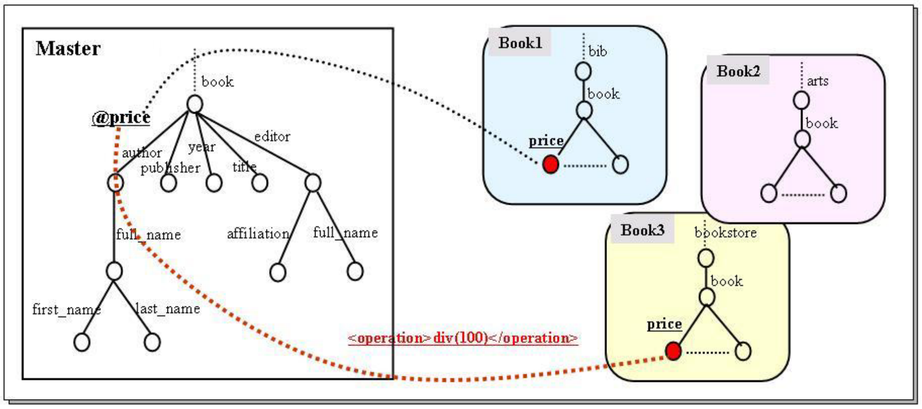
responding elements in local document DTDs are called destination elements. When the query generator finds a source element name in a master query, if its corresponding destination element is not null, then paths in the query are replaced by paths to the destination elements to get a local query (we will see there can be more than one destination element). For example, different sites may represent the author field of the master document as 'author', 'authorname', 'name', 'auth' element, etc. in different local documents. Then the author source element in the DDXMI matches the destination element 'author', 'authorname', 'name', etc., as appropriate for each local document; more complex cases are discussed below.

3.2 Mapping between elements or attributes

Mappings are classified as One-to-One, N-to-One, and One-to-N, according to the number of nodes (including both elements and attributes) involved in the master DTD and a local DTD. There are three different cases for One-to-One mapping: from an element to an element, from an element to an attribute, and from an attribute to an element. Mapping from an element to an element is skipped here, since it is simpler than mapping between an element and an attribute.

One-to-One, from element to attribute There are two cases when an element is mapped to an attribute. If the element is a terminal node, then the element name is simply matched with the attribute name. Otherwise, the attribute name is removed from the path of the local document, since it does not contribute to the path name in the DTD tree.

One-to-One, from an attribute It is straightforward to handle an attribute in the master DTD, because an attribute is generally a leaf node of the master DTD tree. If it matches an attribute, just replace the name. But if an element is matched, that element will appear in the path name even if the attribute name is not in the master query. Figure 3 shows a mapping from a price attribute in the master query to a price element in a local document. The attribute name is replaced by the element name. Notice that, even for mappings between attributes and elements, there can be cases like those we classified in the previous section as 1:N and N:1, and there can also be semantic conversion functions.



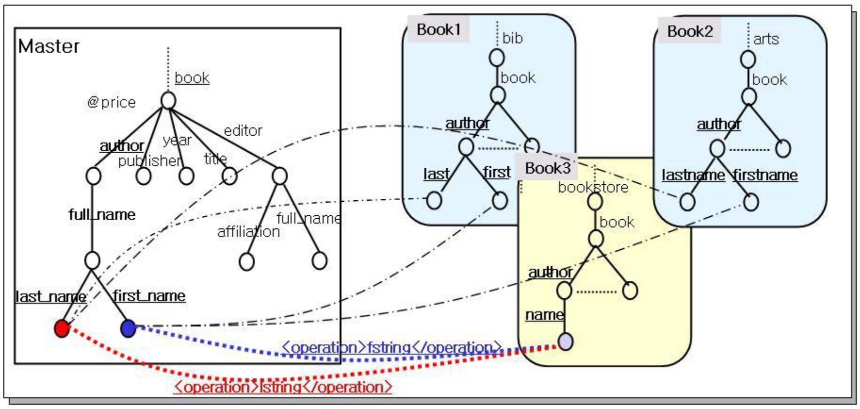
```
<source>/book/@price</source>
    <destination>/bib/book/price</destination>
    <destination>/bookstore/book/price</destination>
```

Fig. 3. One-to-One, attribute to element mapping

Conflicts can arise between representations. For example, the price field in the master DTD uses dollars, but the Book3.DTD in Figure 4 represents it in cents. Some mechanism is required to translate such representations. In this case, when the query is parsed, it is replaced by the result of applying the function `price/100` to convert to the dollar unit.

N-to-One, and One-to-N If two or more nodes of the master DTD correspond to one node in a local document, then the local DTD node will have more than one index number. For example, the first_name and last_name nodes in the master DTD tree in Figure 4 are mapped to the name node in the Book3 DTD tree. In this figure, only the Book3 DTD has full names; the others have separated first name and last name. Hence the separation function names, `fstring` and `lstring`, are included in the DDXMI file for the full name node of the Book3 DTD.

The opposite case is where one node in the master DTD maps to several nodes in a local document. For example, the editor name in the master DTD may be represented with separate first and last names in a local document. The `con` function concatenates the first and the last name elements to get the full name.



```
<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>
```

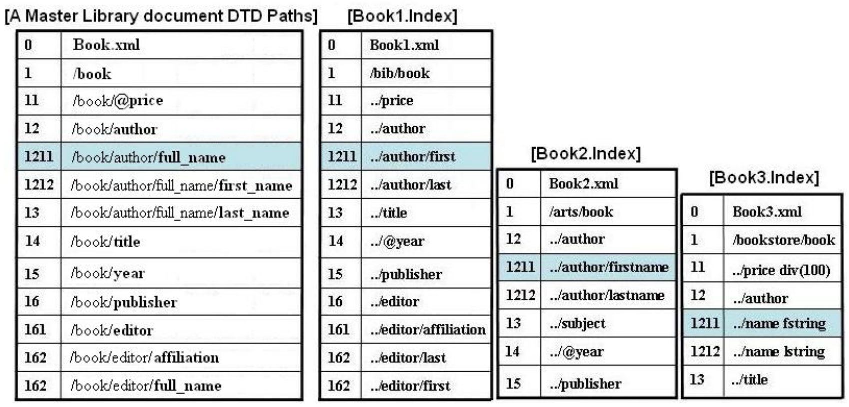
Fig. 4. N-to-One example

3.3 Replacing paths in a query

XML query languages have two kinds of paths, relative and absolute. A path name after '/' is absolute, and one after '//' is relative. A master element has a path name corresponding to local element names if they have 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.

4 Query generation and execution examples

Assume we wish to build a library master document of DTD paths and three local document DTD trees as shown in Figure 5. The same 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 Book2.DTD doesn't have a price element. Based on the index assignment in Figure 5, 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.



[A Master Library document DTD Paths]

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

[Book1.Index]

0	Book1.xml
1	/bib/book
11	../price
12	../author
1211	../author/first
1212	../author/last
13	../title
14	../@year
15	../publisher
16	../editor
161	../editor/affiliation
162	../editor/last
162	../editor/first

[Book2.Index]

0	Book2.xml
1	/arts/book
12	../author
1211	../author/firstname
1212	../author/lastname
13	../subject
14	../@year
15	../publisher

[Book3.Index]

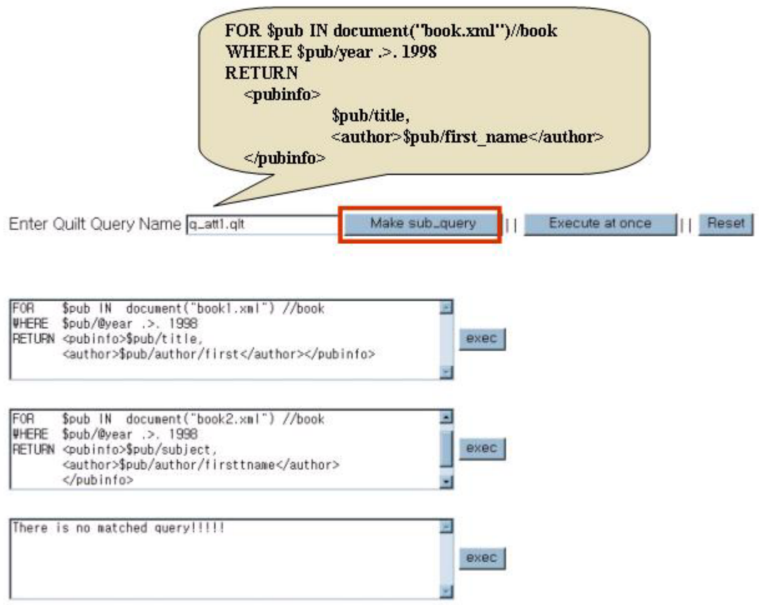
0	Book3.xml
1	/bookstore/book
11	../price div(100)
12	../author
1211	../name fstring
1212	../name lstring
13	../title

Fig. 5. Book1, Book2, Book3 indexed DTD trees, with master DTD

When a user enters a query name, our system generates the local queries. Figures 6 and 7 show examples of queries, their generated local queries, and the result of their execution, for the above three XML documents. Quilt is our XML query language, and Kweelt, developed at the University of Washington, is used to execute Quilt queries.

5 Conclusion and remaining issues

We have built a system for resolving both structural and semantic conflicts in a distributed document system (assuming each local document is in XML format, or the local sources accept XML queries and export XML data) using a GUI tool for generating a file called DDXMI that contains information about data structures and semantics. DTD trees are generated automatically, allowing a data integrator to assign the same index number to nodes with the same meaning and to name semantic functions to resolve representation differences. Then the



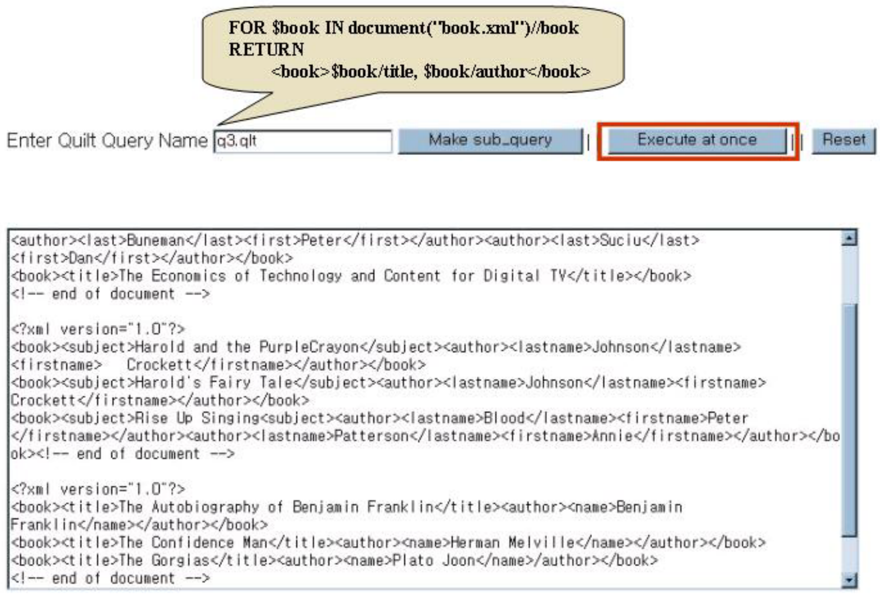
```
FOR $pub IN document("book.xml")//book
WHERE $pub/year .>. 1998
RETURN
   <pubinfo>
               $pub/title,
               <author>$pub/first_name</author>
   </pubinfo>
```

```
FOR    $pub  IN   document("book1.xml") //book
WHERE  $pub/@year .>. 1998
RETURN <pubinfo>$pub/title,
       <author>$pub/author/first</author></pubinfo>
```

```
FOR    $pub  IN   document("book2.xml") //book
WHERE  $pub/@year .>. 1998
RETURN <pubinfo>$pub/subject,
       <author>$pub/author/firsttname</author>
       </pubinfo>
```

```
There is no matched query!!!!!
```

Fig. 6. Example of local query generation



```
FOR $book IN document("book.xml")//book
RETURN
      <book>$book/title, $book/author</book>
```

```
<author><last>Buneman</last><first>Peter</first></author><author><last>Suciu</last>
<first>Dan</first></author></book>
<book><title>The Economics of Technology and Content for Digital TV</title></book>
<!-- end of document -->

<?xml version="1.0"?>
<book><subject>Harold and the PurpleCrayon</subject><author><lastname>Johnson</lastname>
<firstname>   Crockett</firstname></author></book>
<book><subject>Harold's Fairy Tale</subject><author><lastname>Johnson</lastname><firstname>
Crockett</firstname></author></book>
<book><subject>Rise Up Singing<subject><author><lastname>Blood</lastname><firstname>Peter
</firstname></author><author><lastname>Patterson</lastname><firstname>Annie</firstname></author></bo
ok><!-- end of document -->

<?xml version="1.0"?>
<book><title>The Autobiography of Benjamin Franklin</title><author><name>Benjamin
Franklin</name></author></book>
<book><title>The Confidence Man</title><author><name>Herman Melville</name></author></book>
<book><title>The Gorgias</title><author><name>Plato Joon</name>/author></book>
<!-- end of document -->
```

Fig. 7. A query execution result over three documents

DDXMI file is generated by collecting the index numbers. When generating the DDXMI file, the mapping direction is changed from the original local-as-view to global-as-view, to make query rewriting straightforward. A master query from a user is then translated to queries to local documents by looking up the corresponding paths in the DDXMI. Finally the results from local documents are integrated, using the named semantic functions. Our DDXMI system runs under the NT operating system using a Java servlet server and the JavaCC compiler. Any XML documents can be handled without rebuilding or modification. The only requirement for using this approach is to be familiar with the XML data model, and with the relevant documents.

However some issues remain to be investigated. The current mapping mechanism only allows describing is-equivalent mappings between the master paths and local paths. In order to fully use knowledge on the local documents for query decomposition and optimization, it is planned to extend the mapping description power to support describing and using more sophisticated kinds of relationship, and also relationships at more levels, such as local path vs. local path, document vs. document, and document vs. path. Second, the current prototype does not support paths that contain wildcards, but this may happen in a document DTD that includes recursive elements, such as a document for a family tree, or for locations that contain sub-location(s).

If some data does not appear in a local database but is known to the global database, we may speak of 1:0 mapping, and name a semantic function that provides the missing information. An example is when a publisher's database does not include a publisher element. This is not yet implemented, but should not be difficult. More generally, one might consider N:M mappings. If only queries are considered, this is unnecessary, since it decomposes to N mappings that are each 1:M, but if both queries and updates are considered, the concept of N:M mapping is convenient in certain cases.

Another significant issue to be addressed in the future concerns the evolution, or maintenance of metadata about documents and document collections. Even if the documents themselves do not change (which is unlikely if they are really databases, but likely they are really books, pamphlets, magazines, TV shows, etc), it is very possible for our interests and understandings of the documents to change, resulting in different DTDs, which would in turn require generating a new DDXMI. For this reason, it would be desirable to implement an incremental DDXMI tool, that would start from the previous DDXMI file and slightly modify it in light of slight modifications to DTDs, instead of generating it from scratch. Technology for this is well known in computer science, though the implementation would of course require some effort. 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. Management of Data, SIGMOD'99, Philadelphia, 1999.

2. D. Brickley, R. Guha, eds. Resource Description Framework (RDF) Schema Specification. W3C Proposed Recommendation. March 1999. www.w3.org/1999/TR/PR-rdf-schema.
3. K. Beard, T. Smith. A framework for meta-information in digital libraries. In W. Sheth, W. Klas (eds), Multimedia Data Management: Using Metadata to Integrate and Apply Digital Media. McGraw-Hill, pp. 341–365. 1998.
4. Online Computer Library Center, Inc. Dublin Core Metadata Element Set: Reference Description, 1997. Office of Research and Special Projects, Dublin, Ohio. www.oclc.org:5046/research/dublin_core/
5. S. Cluet, P. Veltri, D. Vodislav. Views in a Large Scale XML Repository. 27th VLDB, Roma, Italy, 2001.
6. D. Chamberlin, J. Robie, D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. WebDB 2000, LNCS, Springer, 2000.
7. D. A. Grossman, S. M. Beitzel, E. Jensen, O. Frieder. The IIT Intranet Mediator. IEEE IT Professional, January/February, 2002.
8. A. Gupta, B. Ludascher, M.E. Martone. Knowledge-Based Integration of Neuroscience Data Sources. 12th Intl. Conference Scientific and Statistical Database Management (SSDBM), Berlin, Germany, IEEE Computer Society, July, 2000.
9. D. Egnor, R. Lord. Structured Information Retrieval using XML. In Working Notes of the ACM SIGIR Workshop on XML and Information Retrieval, Athens, Greece. 2000.
10. L.M. Haas, R.J. Miller, B. Niswonger, M. Tork Roth, P.M. Schwarz, E.L. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. Bulletin IEEE Computer Society Technical Committee on Data Engineering. 1999.
11. I. Nishizawa, A. Takasu, J. Adachi. A Schema Integration and Query Processing Information Retrieval. IPSJ SIGNotes DataBase System Abstract 094–009. 1993.
12. A.Y. Levy, A. Rajaraman, J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. VLDB, pp. 251–262, Bombay, India, 1996.
13. R.J. Miller, L.M. Haas, M.A. Hernandez. Schema Mapping as Query Discovery. 26th VLDB. Cairo, 2000.
14. Y.K. Nam, J. Goguen, G. Wang. A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases. LNCS 2519, 2002, pages 1332–1344, Springer. Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems, October 2002.
15. A. Paepcke. An object-oriented view onto public, heterogeneous text databases. In Proceedings Ninth International Conference on Data Engineering, pages 484–493, April 1993.
16. Y. Papakonstantinou, H. Garcia-Molina, J. Ullman. MedMaker: A Mediation System Based on Declarative Specifications. Data Engineering (ICDE). 1996.
17. J.D. Ullman. Information Integration Using Logical Views. 6th Int. Conference Database Theory, LNCS 1186, Springer, 1997.
18. G. Wiederhold. Mediators in the Architecture of Future Information System. IEEE Computer 25:3, pp. 38–49, 1992.
19. XML Metadata Interchange. www-4.ibm.com/software/ad/library/standards/xmi.html. (XMI)