Information Retrieval from Distributed Semistructured Documents Using Metadata Interface

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

[1]Department of Computer Science, Yonsei University, Wonju, Korea
gjchoe@hosu.yonsei.ac.kr, yknam@dragon.yonsei.ac.kr
[2]Department of Computer Science and Engineering, UCSD, La Jolla, CA 92093
{goguen, guilian}@cs.ucsd.edu

Abstract. We describe a method for retrieving information from distributed heterogeneous semistructured documents, and its implementation in the metadata interface DDXMI (Distributed Document XML Metadata Interface). The system generates local queries appropriate for local schemas from a user query over the global schema and shows the result of the generated queries. The three components are designed to generate the local queries: mappings between global schema and local schemas (extracted from local documents if not given), path substitution, and node identification for resolving the heterogeneity among nodes with the same label that often exist in semistructured data. The system uses Quilt as its XML query language. An experiment is reported over three local semistructured documents: 'thesis', 'reports', and 'journal' documents with 'article' global schema. The prototype was developed under Windows system with Java and JavaCC.

1 Introduction

There is much research on integrating distributed heterogeneous data with explicit schemas, which are called structured data. Besides expensive data warehousing, a major focus is virtual integration, i.e., developing portals that allow uniform querying through a global schema to distributed heterogeneous data [12], [14], [18], [21], [25]. A query over the global schema is usually resolved and answered by consulting mappings between the global and local schemas. Semistructured data models emerged as a result of the efforts to extend database management techniques to data with the irregular, unknown, and frequently changing structures that are becoming more and more common as the Internet grows [1], [2], [13]. However, for semistructured data, structural information is not given explicitly, and usually data are created without any restriction on structure, so that it is much more difficult to develop such data processing system. Because the semistructured data have no specific rules or enforcement of the structure, it often happens that elements with the same tag have different

© Springer-Verlag Berlin Heidelberg 2006

structures and contain different information so that a single element in the global schema may correspond to several elements with the same tag and even the same path in an extracted local schema with different mapping types, i.e., 1:1, 1:N, and N:1 mappings.

We designed a system to address this problem and implemented it in a research system for generating local queries over distributed semistructured documents through a metadata interface and the queries are executed on its own local site. It handles semistructured data with additional functionality to extract schemas for XML documents without explicit schema information, as identifying different types of elements with the same tag in our query processing system. The proposed system architecture is shown in Fig. 1. Queries over the global schema are processed based on the mapping information stored in a structured document called DDXMI (for Distributed Documents XML Metadata Interface), which works as an integrated view over all relevant local schemas. The DDXMI file contains the mapping information and functions to be applied to each local document, along with some identification information such as author, date, comments, etc. The system prototype has two parts: the DDXMI Generator for mapping the global schema with local schemas and producing a DDXMI file, and the Query Generator for generating the local queries and answering queries. Our tool parses a document schema or the document itself if its schema is unknown to get the structure of the document, and then generates a dynamic path tree, which can be folded and unfolded by clicking. The mapping is specified by assigning indices through clicking involved nodes in the path trees in a GUI, which link local elements to corresponding global elements and to the names of conversion functions. These functions can be built-in or user-defined in Quilt [7], which is our XML query language. The DDXMI document is then generated by collecting over index numbers, which are internal to the system. User queries are

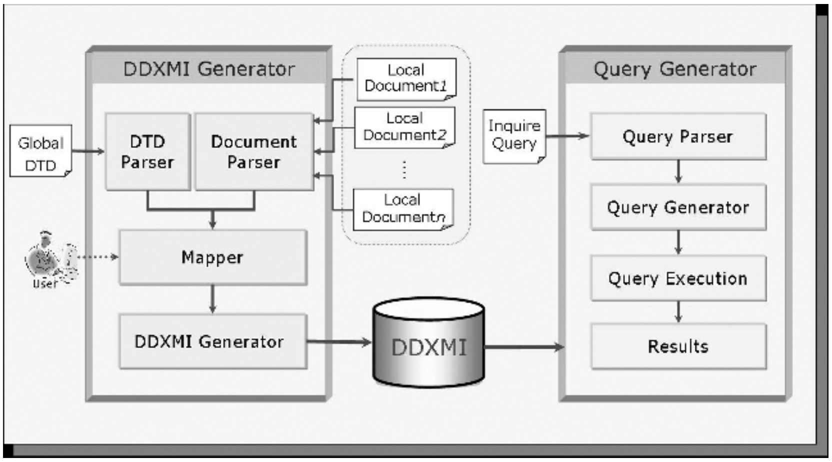


Fig. 1. The structure of the proposed system

rewritten into appropriate queries for each relevant local document according to the mapping information in the DDXMI document and node identification information; finally each local query is processed by Kweelt engine for Quilt.

2 The Related Work

To facilitate formulation, decomposition and optimization of queries for semistructured data, schema extraction or type inference have been studied by using machine learning methods and heuristics [19], [20], [22]. Unfortunately, the accuracy goes down as the extracted schema size decreases.

Schema mapping is a critical step for data integration and many other important database applications. An extensive review of techniques and tools for automatic schema matching up to 2001 is given in [23]. Traditional approaches such as instance-based LSD [8] and GLUE [9] and schema-based Cupid [15], SF [16], Rondo [17] and Coma [10], and the holistic approach MGS [11] only help find 1-to-1 matches, and have great difficulty with matches that involve conditions or conversion functions, and cannot discover n-to-m matches for n>1 or m>1, automatically. Some tools such as COMAP, Clio, SCIA find complex matches based on user input [4], [5], [6], [9], or ontologies [24]. However, it is very difficult for these tools to deal with extremely complex mappings where schema nodes have the same label but different types (these often exist in semistructured data).

Rewriting queries using views has been studied extensively for structured data [12], [18] and for semistructured data [3]. But those researches and techniques all targeted at restricted formats of views.

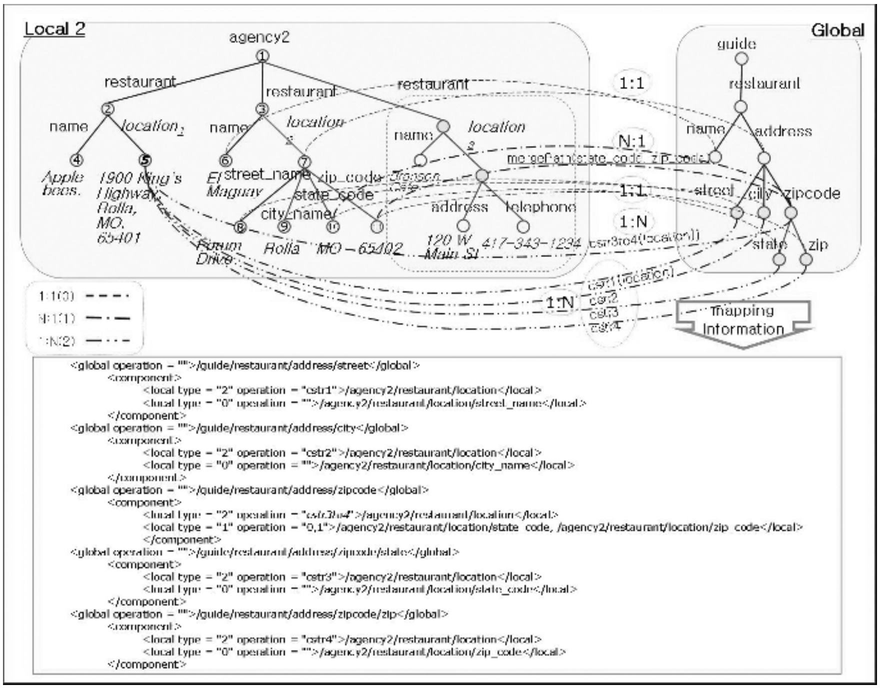
3 The Three Query Processing Components

Our method for generating a set of appropriate local queries Q_{out} from a global query Q_{in} includes three components, *M(LSS, GS), PS*, and *NIP*, where *M* is a component for mapping a global schema *GS* to a set of local schemas *LSS, PS* is a path substitution component, and *NIP* is a node identification predicate generation component for resolving the heterogeneous nodes with the same label, *GS* is a global schema and *LSS* is a set of local schema LS_1 , . . . , LS_j. We describe each of these in the following sub-sections.

3.1 Schema Mapping

The essential part of a system for distributed data sources is the mappings between the global schema and local schemas. Here we describe the semantics of mappings and the mapping representation in our approach.

Assuming that only data are queried and answered from the single document for each site and there are no JOINs among local documents, the total mappings *M(LSS,GS)* are the union of the mappings of the global schema to each of the local schemas, *M(LS,GS)*. Let G and L be the set of nodes in the path tree GT of



```
<global operation = "">/guide/restaurant/address/street</global>
        <component>
                <local type = "2" operation = "cstr1">/agency2/restaurant/location</local>
                <local type = "0" operation = "">/agency2/restaurant/location/street_name</local>
        </component>
<global operation = "">/guide/restaurant/address/city</global>
        <component>
                <local type = "2" operation = "cstr2">/agency2/restaurant/location</local>
                <local type = "0" operation = "">/agency2/restaurant/location/city_name</local>
        </component>
<global operation = "">/guide/restaurant/address/zipcode</global>
        <component>
                <local type = "2" operation = "cstr3to4">/agency2/restaurant/location</local>
                <local type = "1" operation = "0,1">/agency2/restaurant/location/state code, /agency2/restaurant/location/zip code</local>
                </component>
<global operation = "">/guide/restaurant/address/zipcode/state</global>
        <component>
                <local type = "2" operation = "cstr3">/agency2/restaurant/location</local>
                <local type = "0" operation = "">/agency2/restaurant/location/state_code</local>
        </component>
<global operation = "">/guide/restaurant/address/zipcode/zip</global>
        <component>
                <local type = "2" operation = "cstr4">/agency2/restaurant/location</local>
                <local type = "0" operation = "">/agency2/restaurant/location/zip_code</local>
        </component>
```

Fig. 2. Mappings between Global and Local nodes

the global schema GS and the set of nodes in the path tree LT of a local schema LS respectively, and let PG and PL be the power set of G and L respectively. A node o_i in a path tree is an object $(ol_i,\ ov_i)$ which consists of the node label ol_i and the node value ov_i. In Fig. 2, the node number *5* has the node label 'location' and the value '1900 King's Highway, Rolla, MO, 65401'. In GT and LT, several nodes may have the same labels, so we differentiate them by putting the subscript in the label when necessary, such as 'location$_1$' and 'location$_2$'. The mappings $M(LS,GS)$ between the global schema GS and the local schema LS contain mapping elements in the format of $(l,\ g) \in PL{\times}PG$, where $g = (gn_1,\ gn_2,\ \ldots,\ gn_m) \in PG$ and $gn_i \in G$ for $i = 1\ to\ m$ and $l = (ln_1,\ ln_2,\ \ldots,\ ln_n) \in PL$ and $ln_i \in L$ for $i = 1\ to\ n$. We group the mapping elements according to their mapping types as follows:

$$M(LS,GS) = M_{11}(LS,GS) \cup M_{1N}(LS,GS) \cup M_{N1}(LS,GS) \text{ where}$$

i) $M_{11}(LS,\ GS)$ is the set of one-to-one mapping elements $m_{11} = (l,\ g)$, where $g \in PG$ and $l \in PL$ are both the singleton.
ii) $M_{1N}(LS,\ GS)$ is the set of one-to-many mapping elements $m_{1N} = (l,\ g)$, where $g = (gn_1,\ gn_2,\ \ldots,\ gn_m) \in PG,\ m{>}1,\ l \in PL$ is a singleton.
iii) $M_{N1}(LS,\ GS)$ is the set of many-to-one mapping elements $m_{N1} = (l,\ g)$, where $g \in PG$ is a singleton and l is $(ln_1,\ ln_2,\ \ldots,\ ln_n) \in PL,\ n{>}1.$

In Fig. 2, 'guide' and 'agency2' are the names of the global and local schemas respectively, M_{11} *(agency2, guide)* is ((state-code, state), (zip-code, zip)), M_{1N} *(agency2, guide)* is ((location, (street, city, zipcode)), (location, (street, city, state, zip))), and M_{N1} *(agency2, guide)* is ((state-code, zip-code), zipcode).

For $m = (l, g)$ where l is not a singleton, functions are required for combining the content of multiple elements of l into an instance of g. Even for $m = (l, g)$ where l is a singleton, conversion functions are often required for transforming the content of l into an instance of g. We call both combining and conversion functions as transformation functions. Therefore, the transformation T_m over $m = (l, g) \in M(LS, GS)$ is T_m:$l \rightarrow g$ where T_m is a vector of functions applied to the values of objects in l in order to get the appropriate values for objects in g, i.e., $T_m(l) = g$, where $|T_m| = |g|$.

3.2 Path Substitution for Generating Local Queries

Quilt is used as the XML query language in our prototype. A typical Quilt query usually consists of FOR, LET, WHERE and RETURN clauses. FOR clauses are used to bind variables to nodes. In order to identify some specific nodes, more condition may be given inside of '[ ]' predicate. Therefore, path substitution in FOR clauses and WHERE clauses vary according to the mapping kind. In case of N:1 mapping, one global path is mapped by N local paths in a single local document, multiple variables may be introduced for those N nodes, or the parents of the N local nodes are bound and give conditions in predicates. When comparison of node values is involved, relevant transformation functions have to be combined with the paths during path substitution. The primary work for the local query generation from global queries is to replace paths in the global query by the corresponding paths appropriate to the local documents.

For example, in Fig. 2, *PS*(address/zipcode) = (location/zip-code, location/ state-code) since the global element 'address' corresponds to the local element 'location' and the global element 'zipcode' maps to (state-code, zip-code) for many-to-one mapping along the 'location' path, hence *PS*(address/zipcode) = T_m(address/zip-code) = *mergepath*(location/state-code, location/zip-code). *PS* (address/street) = (location) along the 'location' path since there is no mapping for 'address', and 'street' maps to 'location', hence *PS*(address/street) = T_m(address/street) = $cstr_1$(location).

3.3 Resolving the Heterogeneity of Nodes in Local Documents

Recall that the primary difference between structured and semistructured data is that a semistructured document may have several nodes with the same name but different structures. In this case, the nodes with the same label but different structures may map to multiple global nodes in different ways; some may even be mapped and some not, so a condition statement indicating that some unmapped nodes should not participate in the possible candidate answer is needed in the output local query.

For example, in Fig. 2, consider a global query given as Query1 and assume 'address' in 'guide' is only mapped to the nodes 'location$_1$' and 'location$_2$' node and not to 'location$_3$'. The 'location$_3$' node is not relevant to this query. Thus, the local query generator checks whether there are irrelevant nodes to the global query in the local path tree. If so, then such nodes must be explicitly screened by using path filtering predicates.

```
[ Query1 : A global query for 'guide' schema ]
  FOR $addr IN document("guide.xml")//address
  WHERE $addr/zipcode[CONTAINS(.,"MO")]
  RETURN $addr
```

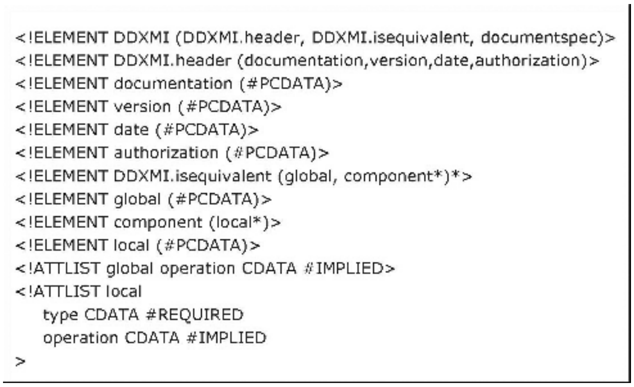
Let l_i and l_j be two nodes with the same label but different structures in a local path tree. If l_i and l_j are mapped to the same global node, then l_i and l_j are called homogeneous, otherwise they are said to conflict. All the nodes sharing the same label with l_i and mapped to the same global node are represented as a set, $homo(l_i)$, while the set of nodes conflicting l_i is $conflict(l_i)$. In Fig. 2, $homo(location_1) = \{location_1, location_2\}$, $conflict(location_1) = \{location_3\}$, and $conflict(location_2) = \{location_3\}$ since 'address' is mapped to 'location$_1$' and 'location$_2$' but not to 'location$_3$'. In Query1, the CONTAINS(.,"MO") predicate is applied to the 'location$_1$' and 'location$_2$' nodes, but not to 'location$_3$' since 'address' maps to only 'location$_1$' and 'location$_2$'. To select the homogeneous elements 'location$_1$' and 'location$_2$', some specific conditions need to be specified.

Let $ln_i \in l$ and $L_{hc}(ln_i)$ be the set of nodes having the same label, but different structure so different index numbers, hence $L_{hc}(ln_i) = homo(ln_i) \cup conflict(ln_i)$. For any element ln_i, $childpaths(ln_i)$ is defined as the set of paths from ln_i's direct children to leaf nodes. The super child path set $SCP(ln_i)$ of ln_i is defined as the set of all child paths for all elements of $L_{hc}(ln_i)$, i.e., $SCP(ln_i) = U_{i=1}^{k}$ $childpaths(h_i)$, where $h_i \in L_{hc}(ln_i)$, $k = |L_{hc}(ln_i)|$. We use $childpaths(ln_i)$ and $SCP(ln_i)$ to formulate predicates to specify only node ln_i while excluding any other nodes sharing the same label with ln_i. The predicate ((p_1 AND, ..., AND p_i) AND (NOT(q_1) AND NOT(q_2), ..., AND NOT(q_j))) for $p_i \in childpaths(ln_i)$ and $q_i \in (SCP(ln_i) - childpaths(ln_i))$ means that ln_i has the child paths $p_1, \ldots, p_i$ and should not have the child paths $q_1, \ldots, q_j$.

4 System Implementation and Execution Examples

4.1 Mapping Representation and Path Substitution

The mapping information for the global schema and local schemas is stored in a structured XML document, a DDXMI file. The structure of DDXMI is specified in DDXMI's DTD, shown in Fig. 3. The elements in the global schema are called global elements, while the corresponding elements in the local documents are called local elements. When the query generator reaches a global element name in a global query, if its corresponding local element is not null, then the paths in the query are replaced by the paths to the local elements to get local queries.



```
<!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 (global, component*)*>
<!ELEMENT global (#PCDATA)>
<!ELEMENT component (local*)>
<!ELEMENT local (#PCDATA)>
<!ATTLIST global operation CDATA #IMPLIED>
<!ATTLIST local
    type CDATA #REQUIRED
    operation CDATA #IMPLIED
>
```

Fig. 3. The DDXMI's DTD

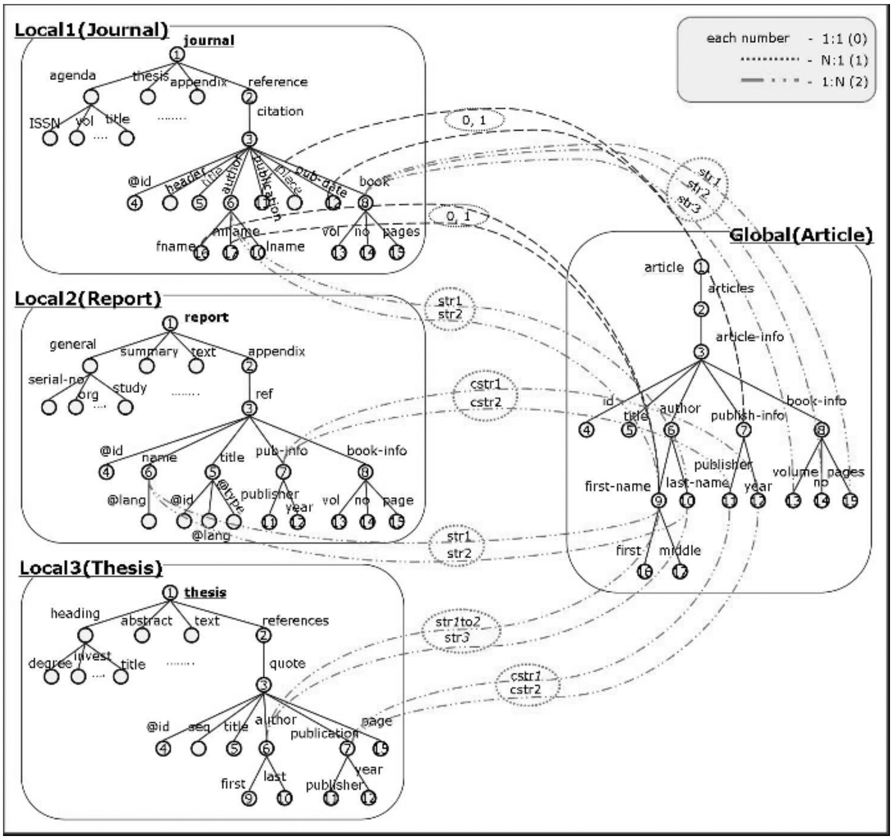
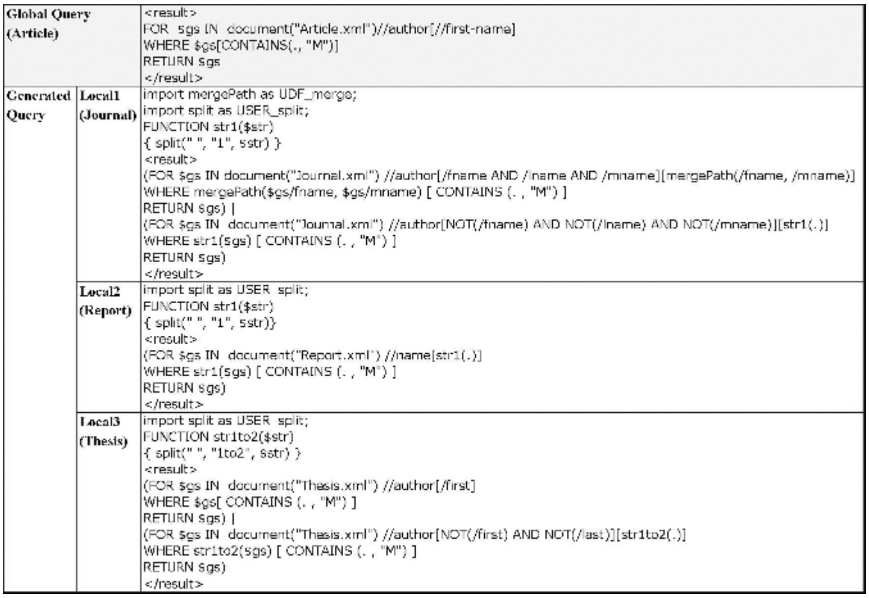


Fig. 4. A portion of the mapping information for 'article' global schema and 3 local documents



Global Query (Article)		`<result>`
		`FOR $gs IN document("Article.xml")//author[//first-name]`
		`WHERE $gs[CONTAINS(., "M")]`
		`RETURN $gs`
		`</result>`

Generated Query	**Local1 (Journal)**	`import mergePath as UDF_merge;`
		`import split as USER_split;`
		`FUNCTION str1($str)`
		`{ split(" ", "1", $str) }`
		`<result>`
		`(FOR $gs IN document("Journal.xml") //author[/fname AND /lname AND /mname][mergePath(/fname, /mname)]`
		`WHERE mergePath($gs/fname, $gs/mname) [ CONTAINS (. , "M") ]`
		`RETURN $gs) |`
		`(FOR $gs IN document("Journal.xml") //author[NOT(/fname) AND NOT(/lname) AND NOT(/mname)][str1(.)]`
		`WHERE str1($gs) [ CONTAINS (. , "M") ]`
		`RETURN $gs)`
		`</result>`

	Local2 (Report)	`import split as USER_split;`
		`FUNCTION str1($str)`
		`{ split(" ", "1", $str)}`
		`<result>`
		`(FOR $gs IN document("Report.xml") //name[str1(.)]`
		`WHERE str1($gs) [ CONTAINS (. , "M") ]`
		`RETURN $gs)`
		`</result>`

	Local3 (Thesis)	`import split as USER_split;`
		`FUNCTION str1to2($str)`
		`{ split(" ", "1to2", $str) }`
		`<result>`
		`(FOR $gs IN document("Thesis.xml") //author[/first]`
		`WHERE $gs[ CONTAINS (. , "M") ]`
		`RETURN $gs) |`
		`(FOR $gs IN document("Thesis.xml") //author[NOT(/first) AND NOT(/last)][str1to2(.)]`
		`WHERE str1to2($gs) [ CONTAINS (. , "M") ]`
		`RETURN $gs)`
		`</result>`

Fig. 5. A global query and the generated local queries

The type attribute in local is for mapping kind; 0, 1, and 2 for one-to-one, one-to-many, and many-to-one respectively; if operation attributes are included, the value of 'operation' attribute is applied to the content of the relevant local nodes in order to get data consistent with the global schema.

The <local> and <global> elements are absolute paths from the root node, which represented as '/', to the leaf nodes. For the example in Fig. 2, the <global> element for 'street' node is '/guide/restaurant/address/street' and its <local> node is '/agency2/restaurant/location'. Therefore, the mapping node for 'street' is 'location' only if the parent node of 'street' node is mapped, otherwise, its mapping node is the difference of the path between the nearest mapped ancestor node and the current node in DDXMI. This means that 'street' is mapped to *strdiff*('/agency2/restaurant/location' - 'agency2/restaurant') = 'location'. Since the mapping type of 'street' node is *2* and the attribute value of its operation is '*cstr1*', it can be easily seen that this is a 1:N mapping, and the transformation function '*cstr1*' is applied to the value of 'location' node, where '*cstr*1' is the name of function separating a string into a set of strings delimited by comma. When the 'street' node is encountered in the parsing process, it is automatically replaced by either the string '*cstr1*(location)' or 'location' depending on the type of the Quilt statement.

The mapping information for N:1 mapping types is stored in DDXMI by separating the node paths by comma. In Fig. 2, the <local> elements of 'zipcode' are '/agency2/restaurant/location/state-code' and '/agency2/restaurant/location/zip-code' and the attribute value of its operation is '0, 1', which is used to indicate the merging sequence. Therefore, 'zipcode' is transformed into '*mergepath*(state-code, zip-code)'.

4.2 Experimental Results

To demonstrate how the system works, we report an experiment with integration of information from 3 local documents: '*thesis*', '*reports*', and '*journal*' semi-structured documents. Assume that we are going to build 'article' database as a virtual global document from information maintained by 'thesis', 'reports', and 'journal' local documents.

The mapping between global and local schemas is shown in Fig. 4. An example Quilt queries getting author's name whose first name contains 'M' letter and the generated local queries from them are shown in Fig. 5.

5 Conclusion and Remaining Issues

A system for generating local queries corresponding to a query of the virtual global schema over distributed semistructured data has been described, with a focus on resolving both structural and semantic conflicts among data sources. It consists of mapping, path substitution, and node identification mechanisms. Especially, it handles the multiple mapping on an element and the node identification among the elements with the same label and different meanings.

The DDXMI file is generated by collecting the paths with the same index numbers. Global queries from end users are translated to appropriate queries to local documents by looking up the corresponding paths and possible semantic functions in the DDXMI, and node identification information. Finally local queries are executed by Kweelt.

There are several obvious limitations with the query processing algorithm and its implementation. Firstly, we extract path trees for documents without explicit schemas using an algorithm that may produce extremely large path trees for irregular semistructured data, which may be too difficult for human to handle. It is desirable to explore how to balance the accuracy and size of approximate typing in practice. Secondly, JOINs among local data are not considered. In order to fully use knowledge of 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.

References

1. S. Abiteboul. Querying semistructured data. In Proceedings of ICDT, 1997
2. Peter Buneman. Tutorial: Semistructured data. In Proceedings of PODs, 1997.
3. Andrea Cal'y, Diego Calvanese, Giuseppe De Giacomo and Maurizio Lenzerini. View-based query answering and query containment over semistructured data. In: Proc. of DBPL 2001.

4. Lucian Popa, Mauricio. Hernandez, Yannis Velegrakis, Renee J. Miller, Felix Naumann, and Howard Ho. Mapping xml and relational schemas with clio. Demo on ICDE 2002.
5. Lucian Popa, Yannis Velegrakis, Renee Miller, Mauricio. Hernandez, and Ronald Fagin. Translating web data. Proc. 28th VLDB Conf., 2002.
6. Joseph Goguen. Data, schema and ontology integration. In Walter Carnielli, Miguel Dionisio, and Paulo Mateus, editors, Proc. Comblog'04, pages 21-31, 2004.T
7. 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.
8. An-Hai Doan, Pedro Domingos and Alon Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. Proc. SIGMOD, 2001.
9. An-Hai Doan. Thesis: Learning to Translate between Structured Representations of Data.University of Washington, 2003.
10. Hong-Hai Do and Erhard Rahm. Coma - a system for flexible combination of schema matching approaches. Proc. 28th VLDB Conf., 2002.
11. Bin He and Kevin Chen-Chuan Chang. Statistical Schema Matching across Web Query Interfaces. Proc. SIGMOD, 2003.
12. A. Y. Levy. Answering Queries Using Views: A Survey. VLDB Journal, 2001.
13. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management systems for semistructured data. SIGMOD Record, 26, 1997.
14. Alon Levy. The Information Manifold approach to Data Integration. IEEE Intelligent Systems, vol.13, pages:12–16,1998.
15. Jayant Madhavan,Philip Bernstein and Erhard Rahm. Generic Schema Matching with Cupid. Proc. 27th VLDB Conference, 2001.
16. Sergey Melnik, Hector Garcia-Molina and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. Proc. ICDE,2002.
17. Sergey Melnik, Erhard Rahm and Philip Bernstein. Rondo: A Programming Platform for Generic Model Management. Proc.SIGMOD,2003.
18. J. D. Ullman. Information integration using logical views. International Conference on Database Theory (ICDT), pages 19-40, 1997.
19. S. Nestorov, S. Abiteboul, and R. Motwani. Inferring Structure in Semistructured Data. In Proceedings of the Workshop on Management of Semistructured Data, 1997.
20. Svetlozar Nestorov, Serge Abiteboul and Rajeev Motwani. Extracting schema from semistructured data. In Proceedings of SIGMOD, pages 295-306, 1998.
21. Y. K. Nam, J. Goguen, and G. Wang. A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases. Springer, LNCS, Volume 2519, pages 1332-1344, 2002.
22. Svetlozar Nestorov and Jeffrey D. Ullman and Janet L. Wiener and Sudarshan S. Chawathe. Representative Objects: Concise representations of Semistructured, Hierarchical Data. Proceeding of ICDE, pages 79-90, 1997.
23. Erhard Rahm and Philip Bernstein. On Matching Schemas Automatically. Technical report, Dept. Computer Science, Univ. of Leipzig, 2001.
24. Li Xu and David Embley. Using Domain Ontologies to Discover Direct and Indirect Matches for Schema Elements. Proc. Semantic Integration Workshop, 2003.
25. Hector Garcia-Molina, Yannis Papakonstantinou, D. Quass, Anand Rajarman, Y. Sagiv, Jeffrey Ullman, Vasilis Vassalos and Jennifer Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. Intelligent Information System, 8(2), 1997.