Mediation-Based XML Query Answerability

Hong-Quang Nguyen[1], Wenny J. Rahayu[1], David Taniar[2], and Kinh Nguyen[1]

[1] Department of Computer Science and Computer Engineering,
La Trobe University, VIC 3086, Australia
{h2Onguyen,wenny,kinh.nguyen}@cs.latrobe.edu.au
[2] School of Business Systems
Monash University, Clayton, VIC 3800, Australia
david.taniar@infotech.monash.edu.au

Abstract. This paper presents a novel mediation-based query answering approach which allows users (1) to reuse their own predefined queries to retrieve information properly from their local data source, and (2) to reformulate those queries in terms of remote data sources in order to obtain additional relevant information. The problem of structural diversity in XML design (e.g. nesting discrepancy and backward paths) makes it difficult to reformulate the queries. Therefore, we highlight the importance of precise query rewriting using *composite concepts* and *relations* of the mediated schema. Our experimental evaluations on real application datasets show that our approach effectively obtains correct answers over a broad diversity of schemas.

1 Introduction

Traditional mediation-based XML query answering approaches allow users to directly pose queries over the global mediated schema; then, the queries are reformulated in terms of each local schema to retrieve answers. We refer it as a *centralized query model*, which suffers several disadvantages. The users at the local data source have to double their effort in learning and working on two different systems: their own local system and the integrated system. Posing the queries over the mediated schema, the users may not be able to find some kinds of data which are very specific to their local data source and are absent on the mediated schema. Further, only a limited number of queries built on the mediated schema are available for the users to obtain their desired answers.

To overcome such disadvantages, we propose a *decentralized query model* in Fig. 1. In this model, the users should still be able to work on their own defined queries on their local schema, increasing the reuse of their queries and reducing the effort of learning the new integrated system. At the same time, they obtain more additional relevant answers from other remote data sources. The users of organization S can use their own familiar *existing* queries Q_S to pose on their own data source S. At the same time, the query is also propagated to the integrated system to ask for extra information from other remote sources. The final answer $A_S \cup (A_{R_1} \cup A_{R_2})$ obtains more answers $(A_{R_1} \cup A_{R_2})$ from R_1 and

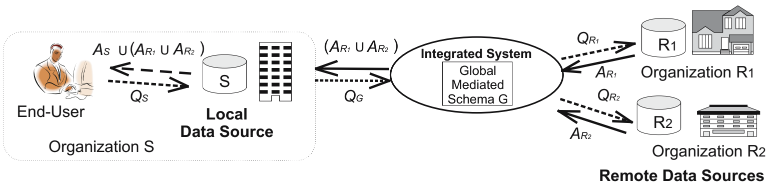

Fig. 1. Decentralized Query Requests

R_2, and combines with desired answer from S. The queries are automatically reformulated, freeing users from the complexity of using multiple systems and making transparent the complex process of query answering.

2 Related Work and Motivation

Several query answering approaches have been proposed in peer data management systems, such as Hyperion [1], PeerDB [2] and Piazza [3]. Peer-based query answering approach can benefit from straightforward pair-wise mappings between two data sources. However, it becomes more complicated when dealing with a large collection of heterogeneous data sources. It may require much human intervention [2] and make the query reformulation process more complicated due to the directional mappings [3]. Mediation-based query rewriting approaches [4,5] have been developed based upon the global schema and mediation mapping rules. However, the global schema is not rich enough to capture semantically hierarchical structure, such as backward paths [5]. To the best of our knowledge, existing approaches mainly use queries available on the global schema rather than reusing the queries available at local data sources. Thus, our approach is novel in the way users pose their queries over the local data source.

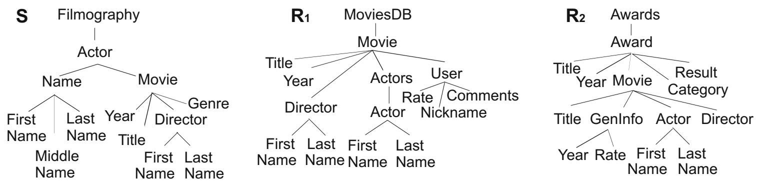

Fig. 2. Case Study: Simplified Schema Repository

Fig. 2 presents a simplified schema repository consisting of three data sources represented by schema trees $D = \{S, R_1, R_2\}$ on *movies* domain. Suppose users from organization S want to search for information from their local data source S, and from remote data sources R_1 and R_2. Consider the following user queries:

Query 1: Find <u>title</u> and <u>year</u> of <u>movie</u>s in which <u>Jackie Chan</u> casts as an <u>actor</u> since <u>2001</u>.

Query 2: Find movies in which Chan is an actor. The detail of the movies include information on directors.
Query 3: Find title of movies in which Chan participates.

With the knowledge of the structure of their local data source S, users may find it simple to write four corresponding XPath queries as follows:

$Q1_S$ = for $m in /*/Actor[Name/FirstName/text()="Jackie" and
 */LastName/text()="Chan"]/Movie[Year >= 2001]
 return <result>$m/Title $m/Year</result>
$Q2_S$ = //Actor[*/LastName/text()="Chan"]/Movie
$Q3_S$ = //Actor[//LastName/text()="Chan"]/Movie/Title

The queries defined above may face a typical problem of *label conflicts* if the remote sources R_1 and R_2 contain different labels conveying the same meaning; e.g. `film` vs. `movie`. The label conflict may arise from label abbreviation or different naming conventions (e.g. `FirstName` or `first-name`). Another problem is *nesting discrepancy* which happens in $Q1_S$ when a node can be represented as either a child or as a decendant of another node but have the same meaning. For example, `Actor` has `FirstName` as a child in R_1 but as a descendant in S. The discrepancy may also result from *backward paths*, such as `Actor/Movie` in S vs. `Movie//Actor` in R_1 and `Movie/Actor` in R_2. Answering query $Q2_S$ will return the target node `Movie` and all of its subtrees; that is, the subtree with root `Director` is implicitly included in the result. Such *implicit inclusion* of a subtree may lead to two main problems: (i) the answer may not include desired data in other remote schemas (e.g. `Director` designed as ancestor or sibling of `Movie`); (ii) the answer may include redundant or irrelevant information such as subtree with root `User` from R_1. Query $Q3_S$ returns movie's title in which `Chan` participates as either an *actor* or a *director*. `Actor` and `Director` have ancestor/descendant relationship in S but have sibling/cousin relationship in R_1 and R_2, making the query answering more difficult. The query reformulation has to identify such contextual meaning for the correct rewriting. Our novel query reformulation approach is proposed to solve all of the problems above.

3 Generating Mediated Schemas

This paper focuses on how to effectively rewrite queries using the global mediated schema, as opposed to the process of creating the global schema. Yet it is important to briefly describe our schema mediation method to make clearer our proposed query answering approach. Frequent subtree mining approach [6] is used to generate the mediated schema. We define *frequency* of a subtree T' of in D, denoted by $f(T')$, is the percentage of the number of trees T in D containing at least one subtree T'. Subtree T' is called *frequent* in D if its frequency is more than or equal to a user-defined minimum support threshold: $f(T') \geq minsup$.

A labeled node in a schema tree represents a concept in the real world. A concept which appears frequently in D is considered to be important and of interest in that domain. Frequent concepts being selected as candidate nodes ensure that only important concepts are retained in the final mediated schema. Omitting infrequent concepts allows us to early remove unimportant or irrelevant concepts. At this step, mediation mappings between constituent schemas and the global schema are retained for the subsequent query reformulation process. We resolve the problem of label conflicts by tokenizing the labels, expanding their abbreviations, and eliminating unimportant parts (such as hyphens and prepositions) [7]. Although frequent concepts found above are of interest, they do not carry much information because they are just single nodes in isolation. They will become more meaningful when they are semantically connected with each other, forming a larger concept, called *composite concept* (*CC* for short).

Definition 1 (Composite/Elementary Concept). *A* composite concept *$X(x_1, x_2, \ldots, x_p)$ is a tree of height 1, which consists of X as the root and $\{x_1, x_2, \ldots, x_p\}$ as a set of X's children. When standing alone, X is referred to as* CC-name*, and x_i ($i = 1..p$) is called* elementary concept *(EC). The following conditions hold:* (X is a non-leaf in D) $\wedge$ (x_i is a leaf in D) $\wedge$ $f(X) \geq minsup \wedge f(x_i) \geq minsup \wedge f(X/\!/x_i) \geq minsup$.

The purpose of a CC $X(x_1, x_2, \ldots, x_p)$ is to bear a coherent semantics representing a real world entity in the domain of interest. The definition on CC also describes what a CC is and how to mine such substructure from our schema sources. For example, `Movie(Title, Year)`, as a whole, is a CC, in which `Movie` is CC-name, and `Title` and `Year` are two ECs. Examining ancestor-descendant path between X and x_i (i.e. embedded subtrees) allows us to solve the problem of nesting discrepancy. It is the structural context of x_i under X that helps clarify the meaning of both X and x_i in the hierarchy. Only such meaningful frequent relationships between two frequent concepts are kept for the construction of the mediated schemas. The mined CCs are currently disconnected from each other. In the real world, the existence of CCs becomes clearer if they interact with each other. The most popular hierarchical path is the forward path because the design of XML structure is generally based on top-down design approach. However, we observe that the same meaning can be represented by `X//Y` or `Y//X`, which are reffered to as *forward* and *backward* paths. Ignoring such semantic similarity will cause information loss. Thus, it is critical to mine both forward and backward paths so that the final mediated schemas become more comprehensive.

Definition 4 (Relation). *Given two CCs $X, Y \in \mathcal{C}$. A* relation *from X to Y, denoted as $X \rightarrow Y$, is defined as a frequent ancestor-descendant path from X to Y. A relation between X and Y can be bidirectional, denoted as $X \leftrightarrow Y$. Let $\mathcal{C}, \mathcal{R}$ denote a set of CCs and a set of relations mined from D, respectively. A relation has to satisfy two rules as follows: (1) Forward paths only: $f(X//Y) \geq minsup \wedge f(Y//X) = 0 \wedge X \rightarrow Y \in \mathcal{R}$; (2) Backward paths: $f(X//Y) > 0 \wedge f(Y//X) > 0 \wedge (X \rightarrow Y \in \mathcal{R} \wedge Y \rightarrow X \in \mathcal{R})$.*

To extract relations, we mine frequent ancestor-descendant $X//Y$ (or $Y//X$) which will be included into the final mediated schema (e.g. `Movie` $\rightarrow$ `Director`. A relation can be either forward pathMining relations based on both forward and backward paths gives a less strict condition by using the sum of the two kinds of paths. This allows more chance of an ancestor-descendant path between X and Y to become frequent; e.g. $\texttt{Actor} \leftrightarrow \texttt{Movie}$.

Definition 5 (Mediated Schema). *Let $\mathcal{C}$ and $\mathcal{R}$ denote a set of mined CCs and a set of mined relations, respectively. A global* mediated schema *is a triple $G = \langle \text{ROOT}, \mathcal{C}, \mathcal{R} \rangle$, where* ROOT *is the root of G.*

We build the mediated schema $G = \langle \mathcal{C}, \mathcal{R} \rangle$, where $\mathcal{C}$=$\{\texttt{Movie}(\texttt{Title}, \texttt{Year}, \texttt{Rate}),$ $\texttt{Actor}(\texttt{FirstName}, \texttt{LastName}), \texttt{Director}(\texttt{FirstName}, \texttt{LastName})\}$, and a set of relations $\mathcal{R} = \{\texttt{Actor} \leftrightarrow \texttt{Movie}, \texttt{Movie} \rightarrow \texttt{Director}\}$, with $minsup = 0.6$. The containment relationship between a CC-name and an CC (such as between `Movie` and `Title`) can be represented by parent-child or ancestor-descendant path in source schemas. Semantic correspondence between constituent schemas and the global schema are captured in the mediation mappings.

4 Query Reformulation Using Mediated Schema

Our approach uses XPath 2.0 syntax [8], a subset language of XQuery, to define the XPath queries. Basically, a XPath expression is defined as a list of nodes and location paths: $p = o_1 n_1 o_2 n_2 \ldots o_k n_k$, where o_i is a location path operator in $\{/, //, *\}$, and $n_i \in TagSet$ is a node in a set of tags. Given an XPath query, we replace variables in predicates and target nodes with values of binding variables in `for` expression. A multi-path query Q_S can be decomposed into different single sub-queries, each of which is reformulated as the normal single-path query.

Next, we decompose all paths in predicates into path expressions and selection conditions. Let $Targ_S$ be a set of paths locating target nodes to be retrieved, $Pred_S$ be a set of predicate paths, and $Cond_S$ contains be a set of conditions in predicates. Path expressions of predicates of source query posed over S are included in $Pred_S$, their corresponding selection conditions are included in $Cond_S$, and path expressions of target nodes are kept in $Targ_S$. Then, we expand all of the abbreviated paths with '*' and '//' in source query Q_S into its equivalent unabbreviated forms. We define a *query pattern* [9] of Q_S as a triple $\langle Targ_S, Pred_S, Cond_S \rangle$. For example, the query pattern of $Q1_S$: $Targ_S =$ $\{/*/\texttt{Actor/Movie/Title}, /*/\texttt{Actor/Movie/Year}\}$, $Pred_S = \{/*/\texttt{Actor/Name}$ $\texttt{/FirstName},/*/\texttt{Actor[Name/LastName},/*/\texttt{Actor/Movie/Year}\}$, $Cond_S =$ $\{\texttt{text()} = \texttt{"Jackie"}, \texttt{text()} = \texttt{"Chan"}, >= 2001\}$.

Given the query pattern of Q_S, we generate the query pattern of Q_G on the mediated schema G by extracting CCs and relations from Q_S. The query pattern of Q_G will be used for the query reformulation on the remote sources in Fig. 3.

a) Identifying Composite Concepts in Source Queries:
We define a function $findCC$: $Pred_S \cup Targ_S \rightarrow \mathcal{C}$ which extracts all CCs from an XPath query by mapping every path in sets of predicates and target nodes into a set of CCs $\mathcal{C}$. Let $p = n_1/n_2/\ldots/n_k$ be a path of k nodes in $(Pred_S \cup Targ_S)$. The last node n_k in path p may belong to one of three cases:

i) n_k is an EC of a CC $n_i(n_k)$: Function $findCC$ first seeks the CC-name n_i to associate with the EC n_k (where $i < k$), and returns the CC $n_i(n_k)$. For example,

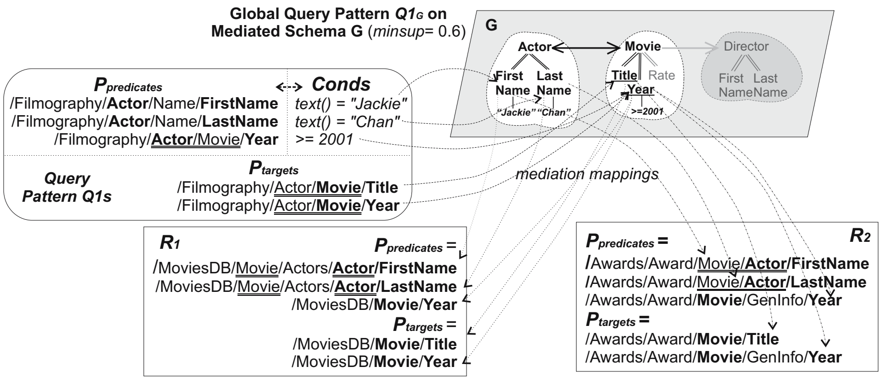
Global Query Pattern Q1G on Mediated Schema G (minsup= 0.6)
G
Actor
Movie
Director
First Name
Last Name
"Jackie" "Chan"
Title
Rate
Year
>=2001
First Last NameName
Ppredicates
Conds
/Filmography/Actor/Name/FirstName
text() = "Jackie"
/Filmography/Actor/Name/LastName
text() = "Chan"
/Filmography/Actor/Movie/Year
>= 2001
Query Pattern Q1s
Ptargets
/Filmography/Actor/Movie/Title
/Filmography/Actor/Movie/Year
mediation mappings
R1
Ppredicates =
/MoviesDB/Movie/Actors/Actor/FirstName
/MoviesDB/Movie/Actors/Actor/LastName
/MoviesDB/Movie/Year
Ptargets =
/MoviesDB/Movie/Title
/MoviesDB/Movie/Year
R2
Ppredicates =
/Awards/Award/Movie/Actor/FirstName
/Awards/Award/Movie/Actor/LastName
/Awards/Award/Movie/GenInfo/Year
Ptargets =
/Awards/Award/Movie/Title
/Awards/Award/Movie/GenInfo/Year

Fig. 3. Query Reformulation Process

the CC `Movie(Title)` is extracted from the target node `/Filmography/Actor/Movie/Title` in the query $Q1_S$.

ii) n_k is a CC-name of a CC in $\mathcal{C}$. If n_k is a predicate node in $Pred_S$, $findCC$ simply returns n_k as a CC-name of a CC in G without returning its ECs. The meaning of the predicate n_k is to check the existence of n_k for returning the answer. Node n_k is actually a non-leaf node which contains a substructure and does not store any value.

If n_k is a target node of the query Q_S, the whole subtree rooted at n_k will be included in the answer. It is important to note that that subtree may include zero or more subtrees (say $n_{k'}$) via implied inclusion, in which $n_{k'}$ corresponds to a CC on G. Thus, $findCC$ will perform two tasks. First, it seeks CC n_k and *all* of its ECs from G. An EC on G may or may not corresponds to any node in S and/or R_i. If the EC does not belong to G, it is considered not of interest to users because it is unpopular and too specific to the local schema S. For example, `/Filmography/Actor/Movie` in $Q2_S$ has the last node `Movie` as a target node which is the CC-name of `Movie(Title, Year, Rate)`; `Rate` is an additional EC that does not appear in S but is included in the answer. Further, $findCC$ extracts the subtree $n_{k'}$ if it exists on the mediated schema G and other remote schemas. In addition to `Movie` found for $Q2_S$, $findCC$ also includes CC `Director(FirstName, LastName)` in the target nodes. Such consideration resolves problem of implicit inclusion.

iii) n_k does not belong to any CC: n_k is neither an EC nor CC-name. This happens when n_k is too specific to the local schema S and is not common among other remote schemas. If n_k is a target node, it cannot be retrieved from other remote sources due to its absence in the global schema G. The query reformulation process will stop without further querying other remote sources. If n_k belongs to a predicate, the query rewritten at G does not contain that predicate. Therefore, the answer returned from the mediated schema (and hence, from remote schemas) is either superset or subset of the desired answer. The identification of CCs in source queries allows non-CC-name nodes to occur between a CC and an

EC. In other words, an EC can be a child node or a descendant node of a CC, addressing the problem of nesting discrepancy.

b) Identifying Relations in Source Queries:
In this section, we identify all relations between CCs from source query Q_S. The relations in Q_S are important to construct path patterns between CCs in queries from remote sources. We define function $findRel : (Preds_S \cup Targs_S) \rightarrow \mathcal{R}$ which maps each path $p \in Preds_S \cup Targs_S$ into a relation $r \in \mathcal{R}$. Function $findRel$ derives relations r between CCs rooted at n_i and n_j from path p based upon the following conditions: (i) $\exists n_i, n_j \in roots(\mathcal{C})$ such that $n_i//n_j \in p$. (ii) $\nexists n_t \in p, n_i, n_j, n_t \in roots(\mathcal{C})$ such that $n_i//n_t//n_j \in p$.

Path p contains more than one node indicating CC (such as n_i and n_j) separated by location paths and non-CC nodes. n_i and n_j forms one relation on p such that there exist no other CC n_t between their path. Relations extracted from source queries establish query pattern Q_G over multiple CCs.

c) Query Reformulation:
Query reformulation involves in using mediated schema to map all paths of source query patterns into their corresponding paths of remote query patterns. Generating mediated schema (section 3) provides mediation mappings between mediated schema and each constituent schema. In other words, we map each element of source query pattern $(Preds_S, Targs_S, Conds_S)$ into a correspondence in remote query pattern $(Pred_{R_i}, Targ_{R_i}, Cond_{R_i})$, respectively. Consider query $Q1_S$. The predicate `/Filmography /Actor/Movie/Year>=2001` of $Q1_S$ is mapped into a conditional EC `Movie(Year)>=2001)` of G, which corresponds to one or more remote query patterns: $Q1_{R_1}$ = `/MoviesDB/Movie/Year>=2001` and $Q1_{R_2}$ = `/Awards/Award/Movie/GenInfo/Year>=2001`. The same process is applied to other paths of source query patterns.

Next, we compose remote query from its query pattern (represented by $Pred_{R_i}, Targ_{R_i}, Cond_{R_i}$). To do this, we merge all of the paths and conditions from its query pattern by defining two generic functions: $prefix$ and $suffix$. Function $prefix$ is defined to map a set of paths $P = \{p_1, p_2, \ldots, p_k\}$ into a single path p' such that p' is the common path (from the root) shared among elements of P. Function $suffix$ returns the suffix of path p_i which is *not shared* with other paths in P. In other words, $suffix(p_i, P) = p_i - prefix(P)$. For example, suppose $P = \{p_1, p_2\} = \{/n_1/n_2/n_3, /n_1/n_3\}$, function $prefix(P)$ returns shared paths between p_1 and p_2, i.e. $prefix(P) = /n_1$ whereas $suffix(p_1, P) = n_2/n_3$ and $suffix(p_2, P) = n_3$.

Definition(Query Answerability). *Given a set of XML documents D, query answerabilty $\mathcal{Q}$ is a measure to determine the ability to find correct answers for a query posed over D, and is defined as the proportion of number of correct answers found to the number of correct answers:* $\mathcal{Q} = \frac{\#Correct\ Answers\ Retrieved}{\#Correct\ Answers}$

The number of correct answers found from D is less than or equal to the number of real answers; thus, $\mathcal{Q}$ is a real number between 0 and 1. The higher value of $\mathcal{Q}$ is, the more correct answers are found.

5 Experimental Evaluation

We perform several experiments on *Movies*[1] – a collection of real applications data collected from Yahoo! Movies (*movies.yahoo.com*) and Internet Movies Databases (*imdb.com*). The Movies dataset contains 1,312 documents of diverse structural designs and a total of 64,706 nodes. The smallest tree has 14 nodes while the largest one contains 91 nodes; on average, there are 49.32 nodes per tree. The height of the trees ranges from 4 to 10 with an average of 6.81.

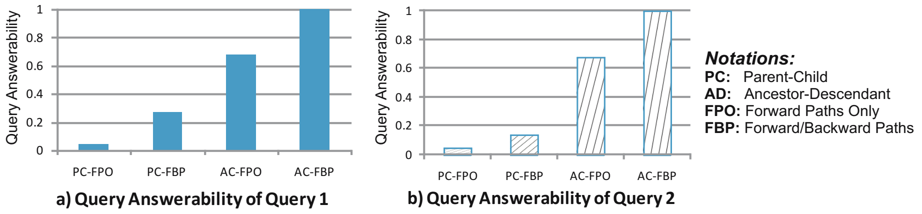

Fig. 4. Query Answerability measures the ability to find correct answers for Query 1-2

We use query answerability to evaluate the quality of different mediation-based query answering approaches. Four different approaches are classified based on the combination of tree traversal types: parent-child (PC) vs. ancestor-descendant (AD), and forward-paths only (FPO) vs. both forward and backward paths (FBP). Fig. 4 presents the query answerability of four kinds of experiments. Among the four experiments, our approach which is based on AD-FBP provides the best query answerability. It obtains complete answers for Query 1 ($Q1_S$) from the dataset with 1.5, 3.67, and 20.9 times improvement compared to PC-FPO, PC-FBP and AD-FPO, respectively. PC-FPO approach returns the worst result with least correct answers for Query 1 and Query 2 (query answerability $\approx$ 0.048) because it limits its search within parent-child paths and forward paths only. When the same concept is expressed as a descendant of a node, the mediated schema based on PC-FPO (e.g. PORSCHE [7]) cannot discover such semantic matching. Hence, its corresponding queries cannot find the correct answers due to the problem of both nesting discrepancy and backward paths. Other traditional schema matching approaches (such as COMA++ [10], Similarity Flooding [11]) only perform pair-wise element matching between two schemas without examining structural context. They do not produce mediated schema for semi-structured documents; thus, they do not belong to any of these approaches. Dealing with nesting discrepancy gives AD-FPO approach an improvement of 2.5 times and 4.8 times the query answerability of Query 1 and Query 2, respectively, based on PC-FBP approach. As a result, query answering based on mediation approaches which resolve both nesting discrepancy and backward paths provides the most comprehensive answers from large collection of heterogeneous XML documents.

[1] http://homepage.cs.latrobe.edu.au/h20nguyen/research

6 Conclusion

In this paper, a mediation-based query reformulation approach is proposed to reuse the existing XPath queries to retrieve more information from other remote sources. It also frees users from the complexity of using multiple systems at the same time. The mediated schema enables us to efficiently reformulate remote queries. Further, it helps prevent source queries with too specific selection conditions from being propagated to remote sources. We resolve the problem of semantic conflicts in labels. Our approach supports users to query and obtains information from heterogeneous data sources of different structures, including nesting discrepancy and backward paths. There are several opportunities for future work. We plan to extend our work to cover other components in structured query languages such as XQuery and XQueryX. Also, we are going to work on the evolution of queries when the global schema evolves.

References

1. Arenas, M., Kantere, V., Kementsietsidis, A., Kiringa, I., Miller, R.J., Mylopoulos, J.: The hyperion project: from data integration to data coordination, vol. 32, pp. 53–58 (2003)
2. Ooi, B.C., Shu, Y., Tan, K.L.: Relational data sharing in peer-based data management systems. SIGMOD Record 32(3), 59–64 (2003)
3. Tatarinov, I., Halevy, A.: Efficient Query Reformulation in Peer Data Management Systems. In: SIGMOD (2004)
4. Halevy, A.Y., Etzioni, O., Doan, A., Ives, Z.G., Madhavan, J., McDowell, L., Tatarinov, I.: Crossing the structure chasm. In: CIDR (2003)
5. Madria, S.K., Passi, K., Bhowmick, S.S.: An xml schema integration and query mechanism system. Data Knowl. Eng. 65(2), 266–303 (2008)
6. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB, pp. 487–499 (1994)
7. Saleem, K., Bellahsene, Z., Hunt, E.: PORSCHE: Performance ORiented SCHEma mediation. Information Systems (2008)
8. Melton, J., Buxton, S.: Querying XML: XQuery, XPath, and SQL/XML in Context. Elsevier, Amsterdam (2006)
9. Yang, L.H., Lee, M.L., Hsu, W., Acharya, S.: Mining Frequent Query Patterns from XML Queries. In: DASFAA 2003 (2003)
10. Do, H.H.: Schema Matching and Mapping-based Data Integration. Ph.D thesis, Dept of Computer Science, University of Leipzig, Germany (2006)
11. Melnik, D., Rahm, E., Bernstein, P.A.: Rondo: A programming platform for generic model management. In: SIGMOD (2003)