Pay-As-You-Go Information Integration: The Semantic Model Approach

Rada Chirkova*
Computer Science Dept.
NC State University
chirkova@csc.ncsu.edu

Dongfeng Chen†
Computer Science Dept.
NC State University
dchen3@ncsu.edu

Fereidoon Sadri‡
Dept. of Computer Science
UNC Greensboro
sadri@uncg.edu

Timo J. Salo
IBM RTP
Research Triangle Park, NC
tjsalo@us.ibm.com

ABSTRACT

This paper describes algorithms for query processing and optimization in our *semantic-model* approach to large-scale information integration and interoperability, and experimentally evaluates the algorithms on real-life and synthetic data. In addition to supporting gradual *(pay-as-you-go)* large-scale information integration and efficient inter-source (join) processing, the semantic-model approach, first described in [34], eliminates the need for mediation in deriving the global schema, thus addressing the main limitation [49] of data-integration systems. The focus of our study in this paper is performance-related characteristics of several alternative approaches that we propose for efficient query processing in the semantic-model environment. Our theoretical results and practical algorithms are of independent interest and can be used in any information-integration system that avoids loading all the data into a single repository. In addition, we present an experimental study of our techniques on real-life and synthetic data. Our experimental results establish the efficiency of our algorithms, and, further allow us to make context-specific recommendations on selecting query-processing approaches from our proposed alternatives. As such, the approaches we propose form a basis for scalable query processing in information integration and interoperability.

1. INTRODUCTION

The need for decentralized data sharing arises naturally in a wide range of applications, including enterprise data management, scientific projects undertaken across universities or research labs, data sharing among governmental databases, and the World-Wide Web. The recent advent of XML as a standard for online data interchange holds much promise toward promoting interoperability and data integration. In addition the focus of information integration has shifted from small-scale integration to providing integration and interoperability among a large number of independent and autonomous information sources. Historically, research and practice in data sharing have focused on data-integration systems, which query distributed shared data through a single central point with a fixed *mediated* schema [52].

Data-integration projects have been a research and commercial success for applications requiring integration of relatively few data sources. At the same time, the need for the mediated schema is a major bottleneck in developing data-sharing products for many real-life applications [49]. Peer data-management systems (PDMS) (*e.g.*, see [24] and references therein) address this limitation by eliminating the need for a mediated schema altogether. In PDMS, each (physical) peer uses its own schema of its stored data and typically interacts with one or more other peers using agreed-on *local* mappings between the respective schemas. This framework makes it easy for a peer to join or leave the peer system at any point in time. Moreover, as some peers in a PDMS may act as mediators *(coordinators)* with respect to other peers, a PDMS can be used as a basis for sharing data on the World-Wide Web, as in the Semantic-Web initiative [4]. See Figure 1 for an example of a PDMS.

PDMS mechanisms for querying the shared data must take into account compositions of the peer-to-peer mappings, which may lead to unsatisfactory query-processing costs in large-scale systems. (Please see [49] for a detailed discussion of the related issues.) In addition, in many practical applications — such as banking, large-scale collaboration among scientific projects undertaken across universities or research labs, data sharing among governmental databases and agencies, and medical information systems — several information sources may store fragments of the same kind of data (conceptually, of the same logical relation), such as information about employees or user accounts in individual bank branches. Coupled with the need for evaluating queries that involve joins of data stored in more than one data source *(inter-source processing),* such data configurations present a further complication in query processing in peer-to-peer systems.

In this paper we address these and other query-processing challenges in large-scale data-sharing systems, by developing algorithms for query evaluation and optimization in our

*Supported by NSF grants Career 0447742 and IIS 0307072.
†Supported by CACC project 07-01 "Integration and Interoperability of XML Data"
‡Supported by NSF grant IIS 0083312.

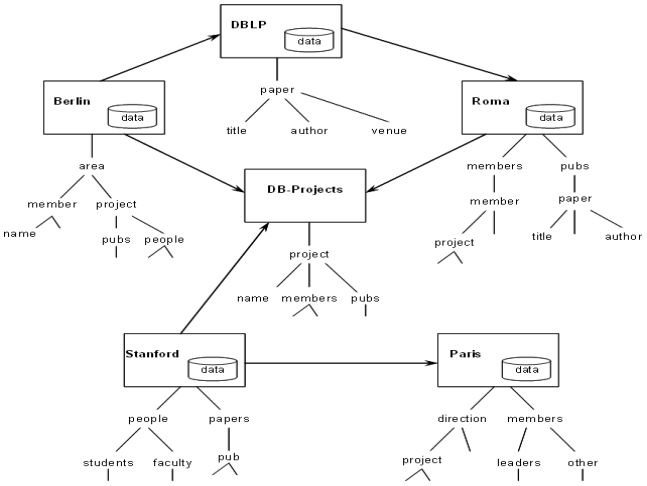

Figure 1: A PDMS for the database research domain. A fragment of each peer's XML schema is shown as a labeled tree (from [49])

semantic-model approach to information integration and interoperability. In the semantic-model (SM) approach, introduced in [34], information at each source is viewed as a collection of (logical) binary relations, which we call the *semantic-model view*. These relations are basically a decomposition of the information into its "atomic components". (For practical simplicity, we allow views that combine binary relations with the same key into a single relation.) The SM approach is consistent with approaches based on ontological modeling [41] in the Semantic-Web initiative, where applications are modeled by the relevant concepts and their properties (basically, by binary relationships), and the semantic-model view for a given source can be designed using an available ontology. To include a source in the integration effort, the source's owner provides mappings from the source's data to the SM view. A source can participate by providing mapping for as few as a single relation, and add more mappings if desired at will. Hence the system supports gradual (pay-as-you-go) integration, where new sources can join incrementally and with small overhead. (This paper is not concerned with the problem of generating the mappings. We refer the reader to [15, 16, 45] for semi-automatic schema mapping techniques, which directly apply to our context.)

Our semantic-model approach differs from existing and proposed information-integration systems. It differs from data-warehousing approaches in that it does not generate a repository of all data, and hence is flexible and can dynamically accommodate sources and scale up to a large number of sources. It differs from information-integration systems that use a mediated schema in that it avoids the lengthy and error-prone process of schema mediation. Intuitively, our SM approach uses ready-made ontologies, or even a much simpler identification of binary relations to serve as an atomic decomposition of the application domain. Again, this allows incremental information integration and scalability. Finally, it differs significantly from peer data-management systems in its capabilities and efficiency in inter-source processing, and its ability to handle sources coming from different application domains. (We provide a more detailed comparison with related projects in Section 5.)

In our SM approach, a user query can be submitted at any site. It can be formulated in terms of the local (original) schema, such as XML or relational, or in terms of the global SM view. The interoperability system is responsible for processing the query. The answers should be as if the query were executed on the overall collection of all data in the participating information sources, while avoiding the generation of a single repository of the data. There are many ways of achieving this goal, with widely varying performances. Query optimization, in particular, is critical to such an interoperability system.

In this paper we address query processing and optimization in our SM approach. The focus of our study in this paper is performance-related characteristics of several alternative approaches that we propose for efficient query processing in the semantic-model environment. One specific focus of our study is reduction or elimination of inter-source processing (namely, of some or even all *inter-source subqueries*) in many practical scenarios. Our theoretical results and practical algorithms are of independent interest and can be used in any information-integration system that avoids loading all the data into a single repository and for combining (merging) XML data. In addition, we present an extensive experimental study of our techniques on real-life data. Our experimental results allow us to make context-specific recommendations on selecting query-processing approaches from our proposed alternatives. As such, the approaches we propose form a basis for scalable query processing in information integration and interoperability.

Our specific contributions:

- We provide a suite of query-processing algorithms for the "subqueries approach" [34] (based on the decomposition of user queries into a number of local and inter-source subqueries), including algorithms for formulating inter-source subqueries and for merging partial XML results.
- We develop a new query-processing algorithm for the SM approach, which we call the *wrapper approach.* Specifically, we propose an algorithm to derive data-source-specific queries that extract from each source the minimum amount of data needed to answer the overall user query. In addition, we experimentally demonstrate that, in many practical cases, an efficient chase-based [51] algorithm can be applied to the union of these intermediate results to produce the answer to the user query.
- We formulate theoretical results that enable us to use key and foreign-key information (*i*) to determine when all inter-source subqueries are redundant, that is, do not produce any new results when compared to the results of local subqueries, and (*ii*) to partition (in case inter-source subqueries are still needed) the subqueries into *required*, *redundant*, and *equivalent sets*, which enable us in our experiments to achieve significantly more efficient query processing in the subqueries approach.
- We propose an algorithm for merging multiple XML data documents with the same schema into a single semantically coherent XML document. This algorithm is of independent interest, as it can also be used to merge multiple overlapping XML documents in a variety of contexts. We use this algorithm in our subqueries approach for merging the results of the subqueries.

- We have implemented the SM approach, with multiple query-processing algorithms, including basic materialization, the subqueries approach, and the wrapper approach. We report the results of our extensive experiments, and establish that our proposed methods are very competitive.
- Our experimental results show that the performance of our methods is query-type and environment dependent, suggesting an (automated) learning approach can be used to establish rules for dynamically selecting the best query processing method for a given query in a specific integration instance.

The remainder of the paper is organized as follows: We provide a summary of our SM approach and discuss efficiency issues and optimization opportunities in Section 2. Our algorithms and theoretical results are presented in Section 3, followed by a discussion of our optimization techniques and experimental results in Section 4. We review related work in Section 5, and conclude in Section 6.

2. THE SEMANTIC-MODEL APPROACH

We begin this section by presenting an overview of the semantic-model (SM) approach, and then concentrate on the optimization challenges and opportunities in this approach. We discuss in Sections 3-4 how we address the challenges and use the opportunities in our proposed algorithms and architectures for the SM approach.

2.1 Overview

The *semantic-model* (SM) approach to information integration and interoperability was first introduced in [34]. In the SM approach, an information source joins an information-integration effort by providing a semantic-model view *(SM view)* of its information, as well as a mapping from its data to this model. The SM view for a source is a view of the information as a collection of binary relations, possibly based on an ontology for the source's application domain. These are basically a decomposition of the information into its "atomic components". (For practical simplicity, we allow views that combine binary relations with the same key into a single relation.) The task of defining the view and providing data mappings is the responsibility of the database administrator (DBA) and/or users of local sources. Tools, such as [38], can greatly simplify the task of defining mapping rules. An SM system contains many *sites*, where a site can be either an information source or a *coordinator* that oversees query decomposition and execution. A user query can be submitted at any site. It can be formulated in terms of the SM view, or in terms of the local (original) schema if submitted at an information source. Currently, our implementation of the SM system supports XML and relational sources, but the SM approach is applicable to any source type as long as mappings from the source data format to the SM view can be provided. The integration and interoperability system is responsible for processing the query in such a way that the answer corresponds to the answer of the query on the collection of all data in the participating information sources. There are many ways to achieve this goal, with widely varying performance characteristics. Query optimization, in particular, is critical to such an interoperability system. The SM approach is consistent with approaches based on ontological modeling [41], where applications are modeled by domain-specific concepts and their properties, which are, basically, binary relationships. The following examples (adopted from [34]) illustrate the SM approach.

EXAMPLE 1 (THE SEMANTIC MODEL). *Consider a federation of catalog sales businesses. In this example we concentrate on their warehousing operations. A possible ontology for this application may use objects* (concepts) *such as* `item`, `warehouse`, `city`, `state`, *and relationships* (properties) *such as* `item-name`, `item-warehouse`, `warehouse-city`, *and* `warehouse-state`. *The SM view consists of binary relations representing the relationships. Sources with heterogeneous models and schemas can model their warehousing operations using this SM view. For example, the DTDs of two XML sources are shown below* [1]. *(We discuss the mappings from these schemas to the SM view in Example 2.)*

```
<!ELEMENT store (warehouse*)>
<!ELEMENT warehouse (city, state, item*)>
<!ELEMENT item (id, name, description)>
<!ATTLIST warehouse id ID #REQUIRED>

<!ELEMENT store (items, warehouses)>
<!ELEMENT items (item*)>
<!ELEMENT item (id, name, description)>
<!ELEMENT warehouses (warehouse*)>
<!ELEMENT warehouse (city, state)>
<!ATTLIST item warehouse-id IDREFS #REQUIRED>
<!ATTLIST warehouse id ID #REQUIRED>
```

The language we use to specify XML-to-SM mappings is based on (a subset of) XPath [53] and is similar to mapping languages, also called "transformation rules" or "source-to-target dependencies" in the literature (see, e.g., [3, 11]). A mapping for a binary relation p has the following general form:

```
p($X, $Y) <- path1 $G, $G/path2 $X, $G/path3 $Y ,
```

where `$X` and `$Y` correspond to the arguments of p. The variable `$G` in the body of the rule is called the "glue" variable, and is used to restrict (`$X`,`$Y`) pairs to have the same `$G` ancestor element in the document.

EXAMPLE 2 (MAPPING RULES). *Consider the first information source of Example 1. Some of the mapping rules that map data in this source to the SM view are as follows:*

```
item-name($I,$N) <-
      /store/warehouse/item $X, $X/id $I, $X/name $N.
item-warehouse($I,$W) <-
      /store/warehouse $X, $X/item/id $I, $X/@id $W.
warehouse-state($W,$S) <-
      /store/warehouse $X, $X/@id $W, $X/state $S.
```

2.2 Efficiency Issues and Optimization Opportunities

For each relation r in the SM view, a source i either stores the information for a *fragment* r_i of r, or has no data relevant to r. The fragment r_i is not materialized; only the mapping rule to generate r_i from the data at source i is available. We assume the data in the system corresponding to a relation r is the union of its fragments at every source, possibly subject to value mappings to reconcile heterogeneities.[2] The answer to a user query Q should reflect

[1] We omit declarations of elements of type `#PCDATA`.

[2] The issue of data heterogeneity, such as different units of measurement, different scales, different terms for the same property, or the same term used for different properties, and ways to handle them are well known and will not be addressed in this paper.

the total data in information sources. That is, if Q mentions relation r, the answer is obtained as if the query were executed on $r = r_1 \cup \ldots \cup r_n$, where $r_1, \ldots, r_n$ are the fragments of r at the information sources in the federation. There are a number of ways to process user queries, with widely different performances. No single scheme is optimum for all queries and cases. Rather, an intelligent query-optimization approach needs to choose from a number of alternatives. We discuss our specific query-processing approaches and algorithms in Section 3, and present experimental results in Section 4. In the remainder of this section we discuss optimization challenges and opportunities in the semantic-model approach.

We envision a *peer/coordinator* system architecture. In the simplest form, all information sources are connected to a single coordinator, which is in charge of coordinating query processing. In general, we can have a network of communicating coordinators, where each coordinator is in charge of a set of sources and, possibly, other coordinators. In fact, a coordinator can be regarded as a source, or, more accurately, as a broker for the information under its oversight. An information source can also function as a coordinator. A pure peer-to-peer system is a special case of a peer/coordinator system where every source is (its own) coordinator.

Consider a user query Q involving k relations $r^1, \ldots, r^k$ in the SM view.[3] Since each relation is the union of its fragments, Q can be regarded as a collection of n^k subqueries, where each subquery corresponds to one combination of single-source fragments of $r^1, \ldots, r^k$. In fact, one way of executing Q is to execute its corresponding subqueries, and then to merge the results. A subquery where all fragments are from the same source is called *local*; such subqueries can be executed at the source with no additional data transfer. A subquery with fragments from two or more sources is an *inter-source subquery*; its execution requires data transfer. Note that out of n^k subqueries, only n are local. The majority $(n^k - n)$ are inter-source subqueries.

The need for inter-source processing arises naturally in information integration. An application may need information from many sources with different kinds of data. For example, a security application may benefit from integrating many sources involving banking, travel, investment, employment, or taxes. There are important queries that need data from many of these sources simultaneously. Even when all sources belong to the same domain, there may be a need for inter-source processing. For example, academic data sources list information about their faculty, students, research, or publications. Each source has complete information about its personnel, but there may be collaboration between groups from different universities. There are queries where the results of local subqueries are only a strict subset of all answers. The need for inter-source processing poses a significant challenge to query optimization in data integration. Blindly executing all subqueries, or, alternatively, materializing the SM view relations by computing their fragments and unioining them, can be very costly. Our approach is based on using known integrity constraints, such as key and foreign-key constraints, to minimize the amount of inter-source processing that is needed. We present a method to determine when no inter-source processing is required to process a given query (Section 3.5). For queries that require some inter-source processing, we present a method to determine the minimum number of subqueries that are required (Section 3.6). These results are of general interest and can be used in any information-integration system that avoids loading all the data into a single repository.

[3]We use superscripts for relations and subscripts for fragments. For example, r_i^j represents the fragment of relation r^j at source i.

3. ALGORITHMS AND THEORETICAL RESULTS

In this section we discuss our main query-processing algorithms and theoretical results that will play a significant role in query optimization. In addition to the basic materialization approach for query evaluation, we describe the *subqueries* and the *wrapper* approaches. Then we discuss theoretical results that allow us (1) to determine when no inter-source processing is needed (Section 3.5), and, (2) when inter-source processing cannot be avoided, to determine minimal equivalence sets of subqueries that are adequate to provide the complete answer to a user query (Section 3.6). These optimization techniques are then discussed in the next section (Section 4), along with the experimental results.

3.1 Query Processing I: Materialization

This is the *base* query-processing approach against which we evaluate other approaches. In the simple *materialization approach,* we materialize the SM view relations that appear in the user query, and execute the query on the materialized relations. Although the approach is not efficient in general, we were surprised to find that this approach was relatively efficient in certain situations, see discussion in Section 4.

3.2 Query Processing II: The Subqueries Approach

The *subqueries approach* is based on generating local and inter-source subqueries for the user query, executing the subqueries, and merging their (partial) results. A *local subquery* uses data from a single source and can be executed at the source. An *inter-source subquery* needs data from multiple sources, and requires some data transmission for its execution. The algorithm for generating local subqueries was presented in [34]. The pseudocode for generating inter-source subqueries is shown in Algorithm 1.

Depending on the key and foreign-key conditions of the semantic-model predicates, we may not need any inter-source subqueries, or may only need a subset of all possible inter-source subqueries. We discuss this issue in detail in Sections 3.5-3.6.

3.3 Query Processing III: The Wrapper Approach

In the *wrapper approach,* we generate one subquery per information source; the subquery extracts from the source the minimum amount of information that is needed to answer the user query. We call this the "wrapper" approach because this extraction can be viewed as a (query-specific) wrapper that collects the needed information from each source. Compared to local subqueries, the information extracted from each source in the wrapper approach is richer than the result of the local subquery on the same source, and makes it possible to obtain the full answer to user query by further processing. In a large class of applications, an efficient *chase*-based algorithm can be applied to the extracted information to obtain the full answer to the user query. We

Algorithm 1: Inter-source subquery generation

input : User query Q, order of data sources, set of binary predicate mappings
output: Inter-source subquery(XQuery) Q'

```
1  foreach predicate in Q do
2      create one variable p for predicate, specifying data
       locations;
3      for each attribute in predicate, create one variable
       using the variable p above;
4  end
5  construct a FOR clause based on the above variables;
6  copy the WHERE clause from Q into Q';
7  replace binary predicates in the WHERE clause with
   the corresponding variables in the FOR clause;
8  foreach head in Q's head elements do
9      generate its RETURN string with the corresponding
       variables in the FOR clause;
10 end
11 concatenate the FOR, WHERE, and RETURN clauses;
12 return XQuery Q' with FLWOR expressions;
```

discuss this approach further in Section 4. The pseudocode for the wrapper algorithm is shown in Algorithm 2.

3.4 Merging XML Data

Given two or more XML documents on the same schema, our *merge algorithm* produces one XML document on the same schema; the document contains all the data from the input documents. In case merging the input documents is not possible, the algorithm outputs the discrepancies that hindered the merge operation. As an example, consider several XML documents of *personnel* information. Certain natural consistency constraints are expected to hold on this kind of data. For example, each individual has a social security number that uniquely identifies the person's name and date of birth. A person may have multiple phone numbers, but the date of birth should be unique. There may or may not be uniqueness constraints on names. The output of running the merge algorithm on such personnel data should contain all individuals mentioned in the input files. In addition, the information for one individual (determined by the same SSN in different inputs) is combined in the output document in the intuitive way: The date of birth of the same individual from different inputs, if known, should be identical. If not, merge is not possible and a discrepancy in the date of birth of this individual is identified. Further, phone numbers from multiple inputs for the same individual are all included in the merged information for that individual. In our prototype implementation, merge is halted if a discrepancy is detected. Many other approaches are possible, including approaches that use information about the degree of reliability of sources to guide the merge in presence of discrepancies, and can be incorporated with relative ease. The pseudocode for the merge algorithm is shown in Algorithms 3 and 4.

3.5 Eliminating Inter-Source Subqueries

In this subsection and in Section 3.6, we present theoretical results that play a significant role in query optimization in the semantic-model approach. Our first result addresses the question "when is inter-source processing not needed?" To motivate this investigation, let us first obtain an intu-

Algorithm 2: The Wrapper Algorithm

input : User query, set of sources *Srcs* and their mappings
output: Single XML document *Doc*

```
1  foreach source in Srcs do
2      create a local subquery for source;
3      execute the subquery locally, then send the local
       result to the coordinator;
4  end
5  merge the local results at the coordinator;
6  if isInterSourceProcessingNeeded() then
7      chaseSteps();
8  end
9  eliminate duplicates in the query answer;
10 save the final answer into XML document Doc;
11 return Doc;

12 Function isInterSourceProcessingNeeded()
13 retrieve binary predicates in the user query;
14 generate a directed graph based on each predicate's key,
   foreign keys, and consistency information;
15 find a direct spanning tree (DST);
16 if failed OR this DST has more than one root then
17     return True;
18 end
19 else
20     return False;
21 end

22 Function chaseSteps()
23 fetch the WHERE clause from the user query;
24 foreach condition in the WHERE clause do
25     if condition doesn't match "predicate-1.attribute-1
       = predicate-2.attribute-2" then
26         continue;
27     end
28     if attribute-1 is the key of predicate-1 then
29         replace nulls at predicate-1 with values at
           predicate-2;
30     end
31     if attribute-2 is the key of predicate-2 then
32         replace nulls at predicate-2 with values at
           predicate-1;
33     end
34 end
35 return the query answer;
```

Algorithm 3: The Merge Algorithm

input : XML documents and their schemas
output: Single XML document D

```
parseSchema(fileStr);
D = mergeAll(dirStr);
Procedure parseSchema(schemaStr)
retrieve keys, unique nodes and other constraints from the schema;
treewalk all elements from root and classify elements into types (single required leaf, single optional leaf, etc...);
Function mergeAll(dirStr)
foreach XML in the directory dirStr do
    DocumentnextDoc = getDoc(XML);
    rsDoc = mergeTwoDocs(rsDoc, nextDoc);
end
write rsDoc into an XML document D;
return D
Function mergeTwoDocs(rsDoc, nextDoc)
get root element root1 from rsDoc;
get root element root2 from nextDoc;
foreach element under root1 do
    mergeElements(element, root2, rsDoc, nextDoc);
end
return rsDoc;
```

ition about the amount of inter-source processing that may be needed: Consider a system with n information sources, and a user query involving k relations in the semantic model. The total number of possible subqueries, where the data for each of the k relations comes from one of the n sources, is n^k. Only n of these are local, in the sense that all data come from the same source. The remaining $n^k - n$ may require some degree of inter-source processing. This is, of course, a worst-case scenario. In practice, even when the total number of sources is very large, a specific relation in the SM view has a limited number of sources with data pertaining to that relation, reducing the possible inter-source queries to $m^k - n$, where $m << n$ is the number of sources with data for a given relation, on the average. Nevertheless, in large-scale information integration, where n can be in the hundreds or even thousands or higher, this number can still be quite large. If we are able to identify the minimum amount of inter-source processing that is required, and restrict our query evaluation to avoid any extra work, we can potentially achieve orders of magnitude faster query processing in large-scale integration.

Our result generalizes a result of [34], which studies user queries that involve the natural join of two relations of the SM view and defines cases when inter-source processing is not needed. We generalize their result to user queries with any number of relations. First, we state the result from [34]:

THEOREM 1. *[34] Consider a user query involving the natural join of relations $r^i(A, B)$ and $r^j(B, C)$. No inter-source processing is needed for this query if all the following conditions hold:*

1. *Key constraint: For every source k, B is the key for the fragment r^j_k.*
2. *Foreign-key constraint: For every source k, there is a foreign-key constraint from $r^i_k(B)$ to $r^j_k(B)$.*

Algorithm 4: The Merge Algorithm (continued)

```
Procedure mergeElements(e1, eleInNextDoc, rsDoc, nextDoc)
get this e1's type typeVal;
switch typeVal do
    case Single Required Leaf
        check existence of the corresponding element e2 in nextDoc;
        compare e1's value with e2's value;
        if either e2 does not exist, or its value is equal to e1's, print ERROR;
        break;
    end
    case Single Optional Leaf
        if e2 exists, compare e1's value with e2's value;
        break;
    end
    case Single Required NonLeaf
        check existence and size of eleInNextDoc in nextDoc;
        ele2 = eleInNextDoc's child;
        foreach element under e1 do
            mergeElements(element, ele2, rsDoc, nextDoc);
        end
        break;
    end
    case Single Optional NonLeaf
        if the size of eleInNextDoc is 1, CALL mergeElements() recursively;
        break;
    end
    case Multi Unique Leaf
        merge eleInNextDoc's children;
        remove duplicates that have the same logical identifiers;
        break;
    end
    case Multi optional Leaf
        merge eleInNextDoc's children, if they exist;
        remove duplicates that have the same logical identifiers;
        break;
    end
    case Multi Unique NonLeaf
        check the unique node in the result of parseSchema();
        CALL mergeElements() recursively;
        break;
    end
    case Multi Optional NonLeaf
    case Multi Reduplicate Leaf
    case Multi Reduplicate NonLeaf
        attach eleInNextDoc's children;
        break;
    end
    case Set Node
        CALL mergeElements() recursively;
        break;
    end
    otherwise
        print ERROR
        break;
    end
end
```

3. Consistency constraint: If $r_k^j(b,c)$ and $r_l^j(b,c')$ hold at two sources k and l, then $c = c'$. ■

In our generalization we use the following definition:

DEFINITION 1. (LOCAL-JOIN GRAPH) Given $r^i(A,B)$ and $r^j(B,C)$, if the three conditions of Theorem 1 hold then we say r^i and r^j have the *local-join property.* Let $r^1,\ldots,r^m$ be all the relations in the SM view. The *local-join graph* is a directed graph $G=(N,E)$, where the set of nodes N corresponds to the relations $r^1,\ldots,r^m$, and $(r^i,r^j)\in E$ if r^i and r^j have the local-join property. ■

Our theorem follows:

THEOREM 2. *Given a user query involving the natural join of two or more relations $r^1,\ldots,r^k$, if the local-join graph restricted to the query relations $\{r^1,\ldots,r^k\}$ contains a directed spanning tree, then no inter-source processing is needed for this query.* ■

In the proof of Theorem 2 we use the following lemma.

LEMMA 1. *If there is an edge from r^i to r^j in the local-join graph, then $(r^1_{s_1}\bowtie\cdots\bowtie r^i_x\bowtie\cdots\bowtie r^j_x\bowtie\cdots\bowtie r^k_{s_k})$ subsumes $(r^1_{s_1}\bowtie\cdots\bowtie r^i_x\bowtie\cdots\bowtie r^j_y\bowtie\cdots\bowtie r^k_{s_k})$ for all s_i, x, and y.*

Proof (Lemma 1): By the local-join property, since r^i has a foreign-key constraint to r^j, then all tuples of the fragment r^i_x at a source x participate in the join with the fragment r^j_x at the same source. Further, by the consistency condition, $r^i_x\bowtie r^j_y$ cannot generate any tuple that is not already in $r^i_x\bowtie r^j_x$. Then, $r^i_x\bowtie r^j_x\supseteq r^i_x\bowtie r^j_y$ for all x and y. Join both sides with $r^1_{s_1}\cdots r^k_{s_k}$. The result of the lemma follows. ■

Proof of Theorem 2:

Let G be the local-join graph of the predicates, and H be the restriction of G to the query relations. Then we want to show that if H has a directed spanning tree T', then the query does not require inter-source processing. Without loss of generality, assume r^1 is the root of T, and $r^1,r^2,\ldots,r^k$ is the depth-first search order of T (any ordering compatible with the parent-child ordering of T will work, including the DFS order). The user query can be written as $r^1\bowtie\cdots\bowtie r^k$. We will show that each inter-source subquery $r^1_{s_1}\bowtie r^2_{s_2}\bowtie\cdots\bowtie r^k_{s_k}$ is subsumed by the local subquery at source s_1, namely $r^1_{s_1}\bowtie r^2_{s_1}\bowtie\cdots\bowtie r^k_{s_1}$. For simplicity, we use $(s_1,s_2,\ldots,s_k)$ to represent the subquery $r^1_{s_1}\bowtie r^2_{s_2}\bowtie\cdots\bowtie r^k_{s_k}$. We now show, by (backward) induction, from $j=k$ to $j=1$, that $(s_1,\ldots,s_1)\supseteq(s_1,\ldots,s_1,s_{j+1},\ldots,s_k)$

Basis $j=k$: Obviously, $(s_1,s_1,\ldots,s_1)\supseteq(s_1,s_1,\ldots,s_1)$.

Induction: Assume the inductive hypothesis holds for $j+1$, that is, $(s_1,\ldots,s_1)\supseteq(s_1,\ldots,s_1,s_{j+1},\ldots,s_k)$. We want to show $(s_1,\ldots,s_1)\supseteq(s_1,\ldots,s_1,s_j,s_{j+1},\ldots,s_k)$. Let r_i be the parent of r_j in the directed spanning tree. Then, since $r_1,\ldots,r_k$ is the depth-first ordering of the tree, we must have $i<j$. By Lemma 1, we have $(r_1^{s_1}\bowtie\cdots\bowtie r_i^{s_1}\bowtie\cdots\bowtie r_j^{s_1}\bowtie\cdots\bowtie r_k^{s_k})$ subsumes $(r_1^{s_1}\bowtie\cdots\bowtie r_i^{s_1}\bowtie\cdots\bowtie r_j^{y}\bowtie\cdots\bowtie r_k^{s_k})$, for all y. Let $y=s_j$. Written in our notation: $(s_1,\ldots,s_1,s_{j+1},\ldots,s_k)\supseteq(s_1,\ldots,s_j,s_{j+1},\ldots,s_k)$. Combined with the inductive hypothesis and by transitivity of subsumption we get: $(s_1,\ldots,s_1)\supseteq(s_1,\ldots,s_j,s_{j+1},\ldots,s_k)$. This completes the proof of the induction.

Hence, the n local subqueries $(s_1,\ldots,s_1)$, $s_1=1,\ldots,n$ (n is the number of sources) subsume all inter-source subqueries. Thus, no inter-source processing is needed in the computation of the user query. ■

Theorem 2 gives a sufficient condition, based on key and foreign-key constraints, where no inter-source processing is needed for the evaluation of a user query. The question naturally arises as to whether the condition of Theorem 2 is also necessary. In other words, if the restriction of the local-join graph to query relations does not have a directed spanning tree, does the evaluation of the query require evaluation of some inter-source subqueries? We should first mention that there are weaker semantic constraints (than key, foreign-key constraints) that may provide conditions for Theorem 2 [34]. However, these semantic constraints are, to the best of our knowledge, not available as standard features in commercial databases. (At the same time, it is possible to enforce such conditions by means of triggers.) Hence, we restrict ourselves to key and foreign-key constraints. This means that if there is no edge from r_i to r_j in the local-join graph, then no constraints of any form exist between r_i and r_j. The following theorem addresses the issue of whether the conditions of Theorem 2 are also necessary in the positive.

THEOREM 3. *Given a user query involving the natural join of two or more relations $r_1,\ldots,r_k$, if the local-join graph restricted to the query relations $\{r_1,\ldots,r_k\}$ does not contain a directed spanning tree, then a database instance exists where at least one inter-source subquery is not subsumed by any local subqueries.* ■

Proof: (Sketch) Let G be the local-join graph restricted to query relations. We show the nodes N of G can be partitioned to 3 subsets: N_1 contains a node r_i and all nodes reachable from r_i; N_2 contains a node r_j that can not reach r_i and is not reachable from r_i, plus all nodes that can reach r_j; and N_3 contains the remaining nodes. We show that N_1 and N_2 are nonempty. Now, we build an instance involving two information sources, where the fragments (1) satisfy the constraints dictated by G, while, (2) there is at least one tuple t in the inter-source join involving relations corresponding to N_1 from source 1, and relations corresponding to N_2 from source 2, such that t is not in the result of any local subquery. Detailed proof can be found at [5]. ■

3.6 Partitioning Inter-Source Subqueries

Given a user query, if the condition of Theorem 2 does not hold then some inter-source processing is needed. In this section we discuss the problem of determining the set of subqueries that are needed for the evaluation of the user query. In particular, we will show a counterintuitive result, namely that the set of needed subqueries is not unique, rather, there can be multiple *equivalence sets* of subqueries. More specifically, we show by a simple example that the set of subqueries can be partitioned into (1) required subqueries, (2) redundant subqueries, with each subquery in this group being subsumed by a subquery in the required group, and (3) zero or more sets of equivalent subqueries, where we need to execute only one subquery from each equivalence class.

EXAMPLE 3. *Consider a system with two sources ($n=2$), and a user query involving the join of three relations $r(A,B)$, $s(A,C)$, $t(A,D)$. Further, assume A is the key for r, and foreign-key constraints hold from $s.A$ and $t.A$ to $r.A$. Also assume the consistency condition (condition 3 in Theorem 1) holds for r. Note that the local-join graph, restricted to r, s, and t, has edges from s and t to r, and does not have a directed spanning tree.*

There are $2^3=8$ subqueries. Two of them are local subqueries, namely, $r_1\bowtie s_1\bowtie t_1$ and $r_2\bowtie s_2\bowtie t_2$ (where r_i

represents the fragment of r that comes from source i, similarly for s and t.) It is easy to verify that (see Section 3.6.1 below), for this query,

- *$r_1 \bowtie s_1 \bowtie t_1$ and $r_2 \bowtie s_2 \bowtie t_2$ are required.*
- *$r_1 \bowtie s_2 \bowtie t_2$ and $r_2 \bowtie s_1 \bowtie t_1$ are redundant: $r_1 \bowtie s_2 \bowtie t_2$ is subsumed by $r_2 \bowtie s_2 \bowtie t_2$, and $r_2 \bowtie s_1 \bowtie t_1$ is subsumed by $r_1 \bowtie s_1 \bowtie t_1$.*
- *$r_1 \bowtie s_1 \bowtie t_2$ and $r_2 \bowtie s_1 \bowtie t_2$ are equivalent, and so are $r_1 \bowtie s_2 \bowtie t_1$ and $r_2 \bowtie s_2 \bowtie t_1$.*

Hence, the user query can be evaluated fully by evaluating four subqueries (out of the total 8). There are four sets of such minimally-sufficient subqueries: Each set includes the two required subqueries, plus one subquery from each of the two equivalence classes in the third bullet above. ■

3.6.1 Determining the partition

We use a graph $G = (N, E)$, which we call the *subsumes graph,* to determine required, redundant, and equivalence classes of subqueries. Each node $q \in N$ represents a subquery. There is also a special node ϕ, intended to represent subqueries with empty results. There is an edge from q_i to q_j if q_i subsumes (i.e., is a superset of) q_j. Further, there is an edge from node ϕ to a node q if the subquery represented by q includes an empty fragment — that is, q represents a subquery involving $r^1_{s_1} \bowtie \cdots \bowtie r^k_{s_k}$ where at least one of the sources s_i does not provide a mapping for the corresponding relation r^i, and hence, $r^i_{s_i}$ is empty. The following procedure can be used to partition the set of subqueries:

- Eliminate all nodes reachable from ϕ (these are the subqueries with empty answers);
- A node with in-degree zero in the remaining graph represents a required subquery;
- Nodes reachable from the required subqueries represent the redundant subqueries;
- Eliminate all nodes representing the required and redundant subqueries; the remaining nodes, if any, must be on cycles; all nodes on a given cycle represent equivalent subqueries.

Finally, how do we determine if one subquery subsumes another? We demonstrate the idea of the algorithm with an example.

EXAMPLE 4. *Consider the same user query and constraints as in Example 3. Consider the edge from s to r in the local-join graph. The following is immediate from the key, foreign-key, and consistency conditions:*
$r_i \bowtie s_i \supseteq r_j \bowtie s_i$, for all $1 < i < n$ and $1 < j < n$ (n is the number of sources). It follows that $r_i \bowtie s_i \bowtie t_k \supseteq r_j \bowtie s_i \bowtie t_k$, for all $1 < i < n$, $1 < j < n$, and $1 < k < n$. Hence, there is an edge from $r_i \bowtie s_i \bowtie t_k$ to $r_j \bowtie s_i \bowtie t_k$, for all $1 < i < n$, $1 < j < n$, $j \neq i$, and $1 < k < n$, in the subsumes graph. ■

4. OPTIMIZATION TECHNIQUES AND EXPERIMENTAL RESULTS

Query-processing performance is a key issue in large-scale information integration. In this section we discuss query-optimization techniques for the semantic-model approach and present experimental results to evaluate the impact of the techniques on the query-processing performance and on the network traffic. We present our experimental results in Section 4.4, after outlining our experimental setup in Section 4.1 and discussing, in Sections 4.2-4.3, the optimization techniques we used in our approach.

Our experimental results show that (a) our methods are competitive with respect to the baseline materialization, and (b) the performance is query-type and environment dependent, suggesting learning approach can be used to establish robust rules or to develop cost functions to be used for dynamically selecting the best query processing method for a given query in a specific integration instance.

4.1 Experimental Setup

In our experiments we used the setup of [49],[4] with the modification that in addition to the queries used in [49] we defined several extra queries that would require more inter-source processing. Our first set of experiments used the "DB-research" dataset, which contains data sources pertaining to several universities, research organizations, and publications information from sources such as CiteSeer [8], DBLP [12], and SIGMOD [48]. Our semantic model for the DB-research dataset, shown in Figure 2, is based on the *academic department ontology* [1] from the DAML ontology library [10]; the queries are shown in Figure 3. (The first ten queries are from [49]; Q11 and Q12 are our extra queries.)

```
[proceeding]
proceeding-title
proceeding-year
proceeding-location
proceeding-gc(general-chair)
proceeding-pc(program-chair)
proceeding-member(program-committee)

[paper]                  [project and person]
paper-title              project-leader
paper-author             project-member
paper-conference         project-paper
paper-cite               project-area
paper-status             project-topic
                         person-adviser
[review]                 person-advisee
review-reviewer          person-affiliation
review-rating            person-homepage
review-paper
review-comment
```

Figure 2: Semantic model for DB-Research

Our second set of experiments used schemas from XML.org. We generated the data for these schemas, since the original data are no longer available on the Internet. As our experiments compare *relative* performance of our algorithms, the comparison is valid no matter what data are used. Our semantic model for the XML.org dataset is shown in Figure 4. The queries are shown in Figure 5; the last query is our extra query.

All the experiments were executed using PostgreSQL [44] version 8.1.3 and SAXON [46] version 8 on a 2.0 GHz Pentium M computer with 768 MB memory and 40 GB hard disk running Windows XP Pro.

[4]We are grateful to Igor Tatarinov and Alon Halevy for providing us with their experimental setup for [49], including the data, schemas, and queries.

```
Q1: Find all XML related projects
Q2: Find all projects a given person was involved in
Q3: Find all co-authors of a given researcher
Q4: Find all papers by a given pair of authors
Q5: Find papers written by researchers who lead an XML
    project
Q6: Find authors who have written reviews about each
    other's papers and who also have co-authored a
    paper
Q7: Find researchers who had a paper at a conference
    where they have been a PC member (perhaps a
    different year)
Q8: Find Jayavel Shanmugasundaram's paper on VLDB'99
Q9: Find PC chairs of recent conferences and their
    papers
Q10: Find pairs of PC-member, submitted paper where
     there is a conflict of interests, i.e., a paper
     author and the PC member have written a paper
     together in the past
Q11: Find people at UC Berkeley and their home pages
Q12: Find students who have published a paper
     co-authored with their supervisors
```

Figure 3: Queries for the DB-research experiments

```
[person]
person-name
person-address
person-phone

[item]
item-description
item-price
item-qty

[order]
order-buyer
order-seller
order-price
order-item
order-itemNum
```

Figure 4: Semantic model for the XML.org schemas

4.2 Reducing Inter-Source Processing

The results presented in Sections 3.5 and 3.6 allow us (1) to determine whether inter-source processing is needed, and, (2) if such processing is needed, to determine equivalent sets of inter-source subqueries that are required and adequate for answering the user query. These results can go a long way toward reducing the overall query-execution costs. Constraints (key and foreign-key) used in these results are quite common and should, in many practical cases, enable us to avoid a large portion of costly inter-source processing. If no inter-source processing is required, the amount of processing needed is $O(n)$, where n is the number of sources, and the processing can be performed locally at each source. Only the last step, merging partial results, needs data transmission. Compare this to a naive execution of all subqueries, where the processing effort is significantly larger — $O(n^k)$, where k is the number of relations in the user query. The amount of data transfer in this case is also significant due to additional data transmission required for inter-source processing.

4.3 The Chase Approach

Our wrapper approach generates for each source local subqueries that, intuitively, extract the minimum amount of information needed to answer the user query. More specifically, if the user query involves the join of relations $r^1, \ldots, r^k$, the wrapper extracts the *outer join* of $r_i^1, \ldots, r_i^k$ from each source i. (Recall that r_i^j denotes the fraction of r^j stored at source i.) With care, selections can also be pushed down to the wrapper. But we have to be careful not to miss any valid answers because of null values. Intuitively, this means we need to treat comparisons with nulls as *true* at the wrapper level, and repeat the comparison once the null has been replaced by a value in our chase process.

An important optimization approach in the wrappers technique is the application of the well-known chase method [51]. In some cases we can obtain the complete answer to the user query by unioning the extracted data, performing a chase with respect to key constraints, enforcing the remaining conditions and projecting on the desired attributes. The chase (with respect to a key constraint, or, more generally, with respect to a functional dependency) can be used to fill in values for nulls in certain cases. The following example illustrates the idea of our algorithm.

EXAMPLE 5. *Consider a user query on the SM view that involves* $r(A, B) \bowtie s(A, C)$*, and assume* A *is the key of* r*. Assume the wrapper generates (among other tuples)* $(a, b, null)$ *from source 1 and* $(a, null, c)$ *from source 2. The result of unioning these two answers and chasing with respect to the key constraint generates the tuple* (a, b, c)*, since the* A*-value* a *is associated with* B*-value* b *in the first tuple, hence the null in the second tuple can be replaced by* b*. Note that* (a, b, c) *is in the (inter-source join)* $r_1 \bowtie s_2$ *and belongs in the answer to the user query.* ■

An efficient sort-based or hashing-based algorithm can be used to implement the chase with respect to functional dependencies. The question arises as to when this simple processing would generate *all* answers to the user query. The answer is, not surprisingly, when the key constraints guarantee the *lossless join* property for the join in the user query.

4.4 Experimental Results

Our experimental results for the DB-Research dataset are shown in Figures 6 and 7, and the results for the XML.org dataset are shown in Figure 8. We show the performance of four algorithms:

- The *materialization* technique generates the relations in the user query at the coordinator, and executes the query on the materialized data.
- The *subqueries* technique executes *all* local and inter-source subqueries, and merges the results at the coordinator.
- The *optimized-subqueries* technique is similar to the subqueries approach, except that it avoids executing redundant inter-source subqueries.
- The *wrapper* technique extracts the minimum required information from each source, combines them at the coordinator, and applies the chase.

The horizontal (X) axis of each of Figures 6-8 lists the queries, and the vertical (Y) axis — the respective execution times in milliseconds. (The 12 queries used in the DB-Research experiments are shown in Figure 3, and the 6 queries used in the XML.org experiments are shown in Figure 5.) Some of the execution times are missing (for Q6, Q7,

```
Q1.  Find all customers
Q2.  Find the parts supplied by a given supplier
Q3.  Find customers of a given supplier
Q4.  Find suppliers who sell a given part
Q5.  Find customers who also supply parts
Q6.  Find customers' ids and phones of a given supplier
```

Figure 5: Queries for the XML.org experiments

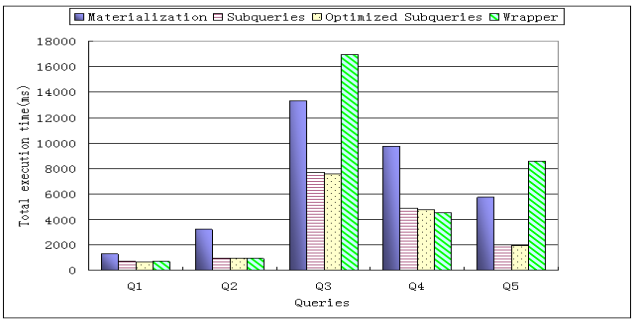

Figure 6: Experimental results for the DB-Research dataset. All query times are given in milliseconds.

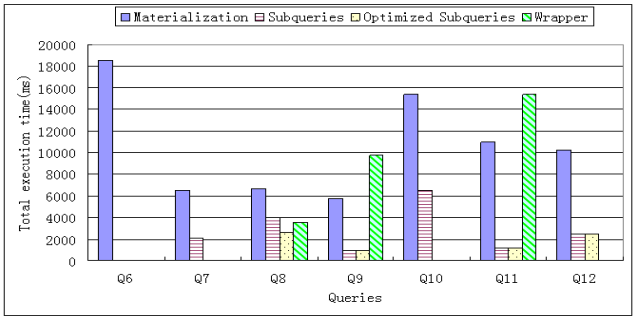

Figure 7: Experimental results for DB-Research (continued). All query times are in milliseconds.

Q10, and Q12 in the DB-Research experiments) due to the SAXON XQuery "out of memory" errors.

The first observation we make is that, although the optimized-subqueries technique seems to be the algorithm of choice for most cases, it is not always the most efficient. In fact, we should not write off any of the algorithms. Rather, we expect that, depending on the information sources and the user query, any of the algorithms may outperform the other three.

The second, more subtle, observation is that our experimental environment may have had an impact on the results, which should be carefully considered when interpreting these results. For XML processing, which included local and inter-source subqueries, and the subqueries for the extraction of minimum required information in the wrapper technique, we used XQuery on SAXON [46]. In fact, the entire wrapper algorithm, including chase and duplicate elimination, is implemented in XQuery. On the other hand, for the materialization technique, we used PostgreSQL [44]. This seems to have penalized the efficiency for the wrapper and, to a lesser degree, for the subqueries approach, while favoring materialization.

We have experimentally established that our proposed methods are very competitive, although, in some cases, the basic materialization method turns out to be the best. The subqueries approach is, in general, more efficient than materialization. The optimized-subqueries technique is more efficient than the other methods for most (but not all) user queries. This is witnessed, in particular, in Q8 of the DB-

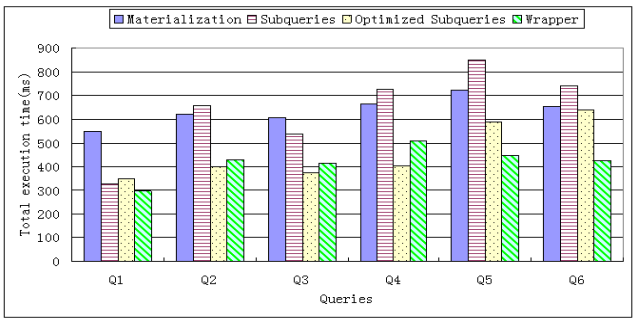

Figure 8: Experimental results for the XML.org dataset. All query times are given in milliseconds.

Research experiments as well as in most XML.org experiments. Despite the handicap of the XQuery-engine inefficiencies, the wrapper technique has the best performance for some of the XML.org experiments. Our conjecture is that when data is very regular at every source, in the sense that each source contributes to all relations in a user query, then the wrapper method is the most efficient. For queries where semantic constraints enable us to eliminate all or most of the inter-source processing, the optimized subqueries is expected to outperform the other methods. Materialization is preferred for queries with large number of relations where all or most inter-source subqueries are required. Finally, subqueries method is preferable for queries with small to moderate number of relations.

Our experimental results show that the performance of our methods is query-type and environment dependent, suggesting an (automated) learning approach can be used to establish robust rules or to develop cost functions to be used for dynamically selecting the best query processing method for a given query in a specific integration instance. The design of such a system is a direction of our ongoing research in this project.

5. RELATED WORK

There has been an increased interest in information integration and interoperability and its applications in recent years, as evidenced by a large number of workshops (such as, Workshop on Information Integration Methods, Architectures, and Systems (IIMAS) (in conjunction with ICDE'08) [29], Workshop on Data Integration in Life Sciences [14], and EDBT 2006 Workshop on Information Integration in Healthcare Applications (IIHA) [28], to name a few). The degree of research activities and publications in information integration and related areas, such as schema matching, model management, and information exchange, have also increased substantially. In this section we will review some of the recent and relevant works.

The importance of large-scale integration (web-scale integration) and "pay-as-you-go" paradigm in such environments have been observed in many recent works, such as [23, 36]. There has been a revitalization of ontological modeling as a result of the W3C Semantic Web initiative [47]. Some information integration systems have been proposed and implemented using ontological modeling concepts. For example, the ICS-FORTH Semantic Web Integration Middleware (SWIM) [6, 7] uses Semantic Web tools for inte-

gration. Main differences with our SM approach is in our use of database tools and concepts, and emphasis on query optimization, versus their reliance on Semantic Web tools, such as RDF and query language RQL [31]. Further, we pay special attention to queries that require data from multiple sources (inter-source processing), while this important issue has not been addressed in SWIM.

In a recent work on indexing large volumes of data [20], authors advocate a *triple* model as a global model for all data. Triples are closely related to RDF and ontologies, and to our SM view model, demonstrating the validity and naturalness of our SM view for modeling the information contents of information sources.

The Piazza and related projects [22, 24, 25, 42, 50] cover various aspects of large-scale data integration, including: (1) Peer-based data management [25, 37, 49, 50], (2) Schema mapping [13, 16, 17, 19, 26, 35], and (3) Theoretical foundations, indexing, and access control [18, 21, 30]. The main idea behind data integration/interoperability in Piazza Peer Data Management System (PDMS) is that users provide mappings between pairs of information sources [25]. There is no need to provide mappings for all pairs. In fact, all that is needed is that the sources graph that represents available mappings be connected. Mappings between any two sources can then be obtained by composing the pairwise mappings along a path connecting the two sources [37, 49]. However, this works well for sources belonging to the same application domain, with similar data. Otherwise the composition process will result in information loss.

Systems based on peer-to-peer mappings are appropriate for federations of information sources with similar information. In contrast, our approach is based on mappings from data sources to simple ontology-based SM views and can be applied to federations of sources with different (as well as similar) information. In our approach mappings are local: No knowledge about other sources is needed. Another advantage of our approach is that the system is more resilient to erroneous mappings (only one source is affected by an erroneous mapping). In contrast, an erroneous mapping in Piazza affects all query processing that uses a path that includes the mapping.

The Clio [9, 40] and Hyperion [2, 27] projects have developed tools for automating common data and structure management tasks underlying many data integration, translation, transformation, and evolution tasks. The thrust of these projects has been on supporting schema management, such as generating, matching, and mapping queries between schemas in multi-source and peer-to-peer systems [33, 39]. Semi-automatic techniques have been developed for general schema matching and for generating mappings between schemas [43, 54]. The projects introduce the *mapping table* approach to represent schema mappings, and discuss query processing in this environment [32, 33]. The architecture is similar to that of Piazza: A query is submitted at a peer, which passes it, possibly in translated form, to (some of) its acquaintances, which repeat this process. In our proposal, schema mappings are always between a simple semantic model and a general schema. We may benefit from some of the discoveries of these projects, but in general, we do not need the sophisticated technologies for general schema matching and mapping.

6. CONCLUSIONS

We presented algorithms for query processing and optimization in our semantic-model approach to large-scale information integration and interoperability. In addition to supporting gradual *(pay-as-you-go)* large-scale information integration and efficient inter-source (join) processing, the SM approach eliminates the need for mediation in deriving the global schema, thus addressing the main limitation of data-integration systems. The focus of our study was performance-related characteristics of several alternative approaches that we proposed for efficient query processing in the semantic-model environment. Our theoretical results and practical algorithms are of independent interest and can be used in any information-integration system that avoids loading all the data into a single repository. We also presented an experimental study of our techniques on real-life and synthetic data. Our experimental results establish the efficiency of our algorithms, and show that the performance of our methods is query-type and environment dependent, suggesting an (automated) learning approach can be used to establish robust rules or to develop cost functions to be used for dynamically selecting the best query processing method for a given query in a specific integration instance. The design of such a system is a direction of our ongoing research in this project.

7. REFERENCES

[1] Academic Department Ontology. `http://www.daml.org/ontologies/65`.

[2] Marcelo Arenas, Vasiliki Kantere, Anastasios Kementsietsidis, Iluju Kiringa, Renée J. Miller, and John Mylopoulos. The Hyperion project: From data integration to data coordination. *SIGMOD Record*, 32(3):53–58, 2003. Special issue on Peer to Peer Data Management.

[3] Marcelo Arenas and Leonid Libkin. XML data exchange: Consistency and query answering. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 13–24, 2005.

[4] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

[5] Dongfeng Chen, Rada Chirkova, and Fereidoon Sadri. Designing an Information Integration and Interoperability System — First Steps. Technical Report NCSU CSC TR-2006-30. Available at `http://www4.ncsu.edu/~rychirko/Papers`, October 2006.

[6] Vassilis Christophides, Gregory Karvounarakis, I. Koffina, G. Kokkinidis, Aimilia Magkanaraki, Dimitris Plexousakis, G. Serfiotis, and Val Tannen. The ICS-FORTH SWIM: A powerful Semantic Web integration middleware. In *Proceedings of VLDB Workshop on Semantic Web and Databases*, pages 381–393, 2003.

[7] Vassilis Christophides, Gregory Karvounarakis, Aimilia Magkanaraki, Dimitris Plexousakis, and Val Tannen. The ICS-FORTH Semantic Web integration middleware (SWIM). In *IEEE Data Engineering Bulletin*, pages 11–18, 2003.

[8] CiteSeer. `http://citeseer.ist.psu.edu/`.

[9] The Clio project. `http://www.cs.toronto.edu/db/clio`.

[10] DAML Ontology Library. `http://www.daml.org/ontologies/`.

[11] Susan Davidson, Wenfei Fan, Carmem Hara, and Jing Qin. Propagating XML constraints to relations. In *Proceedings of IEEE International Conference on Data Engineering*, 2003.

[12] dblp. `http://www.informatik.uni-trier.de/ ley/db/index.html`.

[13] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Y. Halevy, and Pedro Domingos. iMAP: Discovering complex semantic matches between database schemas. In

Proceedings of ACM SIGMOD International Conference on Management of Data, pages 383–394, 2004.

[14] Data Integration in Life Sciences, 2007. http://dils07.cis.upenn.edu.

[15] Hong Hai Do and Erhard Rahm:. COMA - a system for flexible combination of schema matching approaches. In *Proceedings of International Conference on Very Large Databases*, pages 610–621, 2002.

[16] AnHai Doan, Pedro Domingos, and Alon Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2001.

[17] AnHai Doan, Pedro Domingos, and Alon Halevy. Reconciling schemas of disparate data sources: A multistrategy approach. *Machine Learning*, 50(3):279–301, 2003.

[18] AnHai Doan and Alon Halevy. Efficiently ordering query plans for data integration. In *Proceedings of IEEE International Conference on Data Engineering*, pages 393–402, 2002.

[19] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the International WWW Conference*, pages 662–673, 2002.

[20] Xin Dong and Alon Halevy. Indexing dataspaces. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 43–54, 2007.

[21] Xin Dong, Alon Y. Halevy, and Igor Tatarinov. Containment of nested XML queries. In *Proceedings of International Conference on Very Large Databases*, pages 132–143, 2005.

[22] Alon Halevy, Oren Etzioni, AnHai Doan, Zachary Ives, Jayant Madhavan, Luke McDowell, and Igor Tatarinov. Crossing the structure chasm. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2003.

[23] Alon Halevy, Michael Franklin, and David Maier. Principles of dataspace systems. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 1–9, 2006.

[24] Alon Y. Halevy, Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu, and Igor Tatarinov. The Piazza peer data management system. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):787–798, 2004.

[25] Alon Y. Halevy, Zachary G. Ives, Peter Mork, and Igor Tatarinov. Piazza: data management infrastructure for semantic web applications. In *Proceedings of the International WWW Conference*, pages 556–567, 2003.

[26] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation in peer data management systems. In *Proceedings of IEEE International Conference on Data Engineering*, pages 505–516, 2003.

[27] The Hyperion project. http://www.cs.toronto.edu/db/hyperion.

[28] EDBT 2006 Workshop on Information Integration in Healthcare Applications (IIHA), 2006. www6.informatik.uni-erlangen.de/events/wsIIHAedbt2006/.

[29] Workshop on Information Integration Methods, Architectures, and Systems (IIMAS), in conjunction with ICDE'08, 2008. http://daks.ucdavis.edu/IIMAS08/.

[30] Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. Adapting to source properties in processing data integration queries. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 395–406, 2004.

[31] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. RQL: A declarative query language for RDF. In *Proceedings of the International WWW Conference*, pages 592–603, 2002.

[32] Anastasios Kementsietsidis and Marcelo Arenas. Data sharing through query translation in autonomous sources. In *Proceedings of International Conference on Very Large Databases*, pages 468–479, 2004.

[33] Anastasios Kementsietsidis, Marcelo Arenas, and Renée J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 325–336, 2003.

[34] Laks V. S. Lakshmanan and Fereidoon Sadri. Interoperability on XML data. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 146–163, 2003.

[35] Jayant Madhavan, Philip A. Bernstein, AnHai Doan, and Alon Y. Halevy. Corpus-based schema matching. In *Proceedings of IEEE International Conference on Data Engineering*, 2005.

[36] Jayant Madhavan, Shirley Cohen, Xin Luna Dong, Alon Y. Halevy, Shawn R. Jeffery, David Ko, and Cong Yu. Web-scale data integration: You can afford to pay as you go. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 342–350, 2007.

[37] Jayant Madhavan and Alon Y. Halevy. Composing mappings among data sources. In *Proceedings of International Conference on Very Large Databases*, pages 572–583, 2003.

[38] Douglas Markland. An XML mapping tool. Project Report, Department of Mathematical Sciences, University of North Carolina at Greensboro, April 2004.

[39] Renée J. Miller, Laura M. Haas, and Mauricio A. Hernández. Schema mapping as query discovery. In *Proceedings of International Conference on Very Large Databases*, pages 77–88, 2000.

[40] Renée J. Miller, Mauricio A. Hernández, Laura M. Haas, Ling-Ling Yan, C. T. Howard Ho, Ronald Fagin, and Lucian Popa. The Clio project: Managing heterogeneity. *SIGMOD Record*, 30(1), 2001.

[41] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology, 2001. http://ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf.

[42] The Piazza project. http://data.cs.washington.edu/p2p/piazza.

[43] Lucian Popa, Yannis Velegrakis, Renée J. Miller, Mauricio A. Hernández, and Ronald Fagin. Translating web data. In *Proceedings of International Conference on Very Large Databases*, 2002.

[44] PostgreSQL. http://www.postgresql.org/.

[45] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[46] SAXONICA XSLT and XQuery Processing. http://www.saxonica.com/.

[47] Semantic Web. http://www.w3c.org/2001/sw/.

[48] SIGMOD. http://www.sigmod.org/.

[49] Igor Tatarinov and Alon Halevy. Efficient query reformulation in peer data management systems. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 539–550, 2004.

[50] Igor Tatarinov, Zachary G. Ives, Jayant Madhavan, Alon Y. Halevy, Dan Suciu, Nilesh N. Dalvi, Xin Dong, Yana Kadiyska, Gerome Miklau, and Peter Mork. The piazza peer data management project. *SIGMOD Record*, 32(3):47–52, 2003.

[51] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.

[52] Jeffrey D. Ullman. Information integration using logical views. In *Proceedings of International Conference on Database Theory*, pages 19–40, 1997.

[53] XML Path Language (XPath). http://www.w3c.org/TR/xpath.

[54] Ling-Ling Yan, Renée J. Miller, and Laura M. Haas. Data-driven understanding and refinement of schema mappings. In *Proceedings of International Conference on Very Large Databases*, 2001.