GETHERS II, MALCOM B., M.S. Improving Efficiency in XML Interoperability. (2007)
Directed by Dr. Fereidoon Sadri. 67 pp.

The problem of interoperability among XML data sources is one studied by Laks V. S. Lakshmanan and Fereidoon Sadri, and expression their research findings and ideas appear in the paper [7]. Their research presents a lightweight framework that enriches local sources with semantic declarations to enable interoperability.

In their framework, there are issues which reduce the scalability of the architecture, and this thesis identifies those issues and present techniques to overcome boundaries currently placed on their work. The research included in this thesis focuses, first on a methods of obtaining optimal data transmission schemes. Those efforts are followed by developing a new technique to eliminate the costly approach involving inter-source subqueries in [7].

IMPROVING EFFICIENCY IN XML INTEROPERABILITY

by

Malcom B. Gethers II

A Thesis Submitted to
the Faculty of The Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Greensboro
2007

Approved by

Committee Chair

APPROVAL PAGE

This thesis has been approved by the following committee of the
Faculty of The Graduate School at The University of North Carolina at Greensboro.

Committee Chair ______________________________

Committee Members ______________________________

____

____
Date of Acceptance by Committee

____
Date of Final Oral Examination

ACKNOWLEDGMENTS

First and foremost I like to thank God for blessing me with the opportunity and seeing me all the way through. I would especially like to thank Fereidoon Sadri for his guidance and hours of dedication throughout the preparation of this thesis. I would also like to thank the thesis committee members Francine Blanchet-Sadri and Lixin Fu for taking time out of their summers to review this work. Additionally, I would like to thank all of the faculty at the University of North Carolina at Greensboro. I believe that all of the faculty have contributed in some way to helping me complete this thesis.

I would like to thank all my family and friends who supported me and took time out to review this work. I would especially like to thank my friend and love Laura for all of her support and encouragement throughout the years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The notion of data integration represents a long standing open problem in the database community. The research area of data integration has broadened, now extending its focus to Peer to Peer (P2P) approaches to integration [3], data integration for medical research [5], and more. With the expansion of techniques to data integration and application domains, researchers remain motivated to innovate new ideas to bring to the community.

With regards to data integration, Extensible Markup Language (XML) partakes a prominent role in research efforts [4]. This emerged with [1] showing the benefits of semi-structured data. XML's self documenting and human readable properties provide users with flexability not apparent in the traditional relational database systems. Attempts are made by the data integration community to harness those properties in its research.

This thesis builds on a foundation established by *XML Interoperability*. The authors of the aforementioned paper attempt to leverage the developments of the semantic web. In doing so the ability to enrich local sources with semantic declarations becomes available which provides the technology required to compose queries over a application specific common vocabulary of predicates. Expansion of one of the aforementioned queries results in $\bigcirc(n^m)$ subqueries with all but n of those queries requiring data transmission, and the research focuses on providing efficiency in this area of the framework. Discussion of complete details about the framework presented in [7] take place in Section 1.1.

The concept of large scale data integration provided motivation for research efforts herein. [7] presents the concept of inter-source subqueries in its research. These subqueries require data of multiple sources to reside at a central location in order for the evaluation to take place. Obtaining a central location requires the transmission of data from sources required by the inter-source subquery to a location where inter-source processing can take place. Also, this thesis provides discussion about the concept of data transmission. Herein discussion of the notion of optimal data transmission also takes place. From that the research presents a naive approach to acquiring the optimal solution and some other potentially useful concepts regarding efficient data transmission.

Inter-source subqueries cause additional overhead in the framework. In regards to large scale data integration, $\bigcirc(n^m)$ queries could be extremely costly and lead to major system performance issues. This is not just the overhead involved in processing $\bigcirc(n^m)$ queries, but also the cost of transmitting data required for query processing to take place. This leads to the question of whether the elimination of all inter-source subqueries is possible without the framework losing functionality. The research leads to the discovery of an approach that eliminates the need of any inter-source subqueries which is a technique known as the *relational* approach. With the ideas regarding this approach,it may be possible to move closer to establishing a framework with improved scalability, capable of efficient large scale data integration.

1.1 Prior Work

The integration of multiple collections of data should be a simple process. Users should posses the ability to gather results of queries based not only on their own data, but also the data of other sources containing information from a common domain. Doing this should not require knowledge of the precise structure of other users' data. A method, proposed by Laks V. S. Lakshmanan and Fereidoon Sadri, to accomplish the task surfaces in the paper *XML Interoperability.* Neither the problem at hand nor a solution to the problem is unique. The authors present a different technique to address the problem. XML (Extensible Markup Language) dispenses a means of presenting each user's data (also referred to as source). XML's flexibility, constant advancements, and recent establishment as a World Wide Web Consortium (W3C) standard all made it an ideal selection. An overview of the proposed solution is as follows. Multiple users, possibly at different locations, all retain their individual collections of data. All the collections of data partake in a common domain. Therefore, the ability to map all sources to a common vocabulary is possible. This possibility supports method of enabling interoperability amongst multiple XML data sources presented by [7].

The architecture proposed by [7] appears in Figure 1.1. For the processing of a query in the framework, the coordinator first receives and processes the user query in terms of the semantic model. The processing of the query eventually leads to the creation and transmission of local subqueries. Semantic enrichment occurs, allowing individual sources to provide their data. Sources' data (local intermediate results) are returned to the coordinator for any processing which remains. The data from all individual sources now possess the schema of the global predicates. Upon completion the union of all the answers is returned to the user. The framework provides a solid foundation for research efforts in the realm of data integration. This section expands on the concepts of the framework and prepares readers for the new ideas and concepts established by research efforts made during the process of writing this thesis.

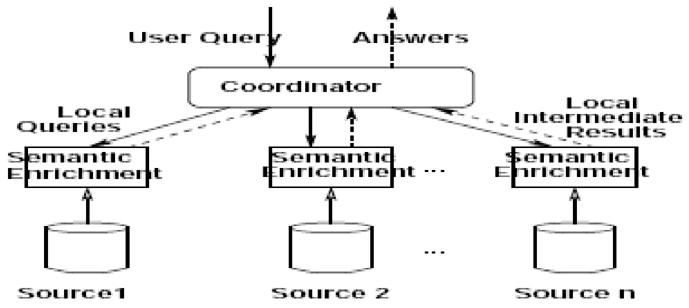

Figure 1.1: Architecture for Interoperability

While providing insight on *XML Interoperability*, a simple example is used to relate key points to a possible real world scenario which pertains to student and course information maintained by institutions. A Document Type Definition (DTD) representing the University appears in Figure B, and this “schema“ aids in illustrating different concepts presented by the paper [7].

Enriching local sources with semantic declarations constitutes the foundation for the lightweight framework which enables interoperability. An ”application specific ontology” provides the source of such declarations (in line with Resource Description Framework (RDF), brought about by the Semantic Web). The critical endeavor of predicate simplification occurs during the creation of the aforementioned ontologies. Creating simple binary predicates is key to the concept at hand.

To provide maximal flexibility predicates should be as simple (not divisible any further) as possible. Establishing that simplicity provides many benefits. Simplicity of predicates alleviate difficulties associated with adding attributes to a global schema. Additionally, it increases the likelyhood that predicates will exist that correspond to data in sources schemas. Binary predicates alleviate heavyweights previously associated with data integration and more specifically the notion of global schema. An ontology corresponding to the presented example includes `id-name`, `id-major`, `id-cno`, `cno-title`, and `cno-credit`. The arguments `id` and `cno` correspond to an individual student and a particular course respectively. The ontology also contains detailed information pertaining to each argument of the ontology. Take note to the fact that each predicate illustrates a relationship between two arguments.

The use of the simplified binary predicates allows users to selectively create mappings for particular predicates. This allows users who lack information for all predicates to still contribute data. Furthermore, it provides the user discretion about what information they wish to share. If the ontology contains predicates which require data the source considers no disclosable, the source has the option of not mapping to those predicates.

Binary predicates gain their usefulness upon the establishment of connections between the predicate and individual sources. This occurs through the mapping of

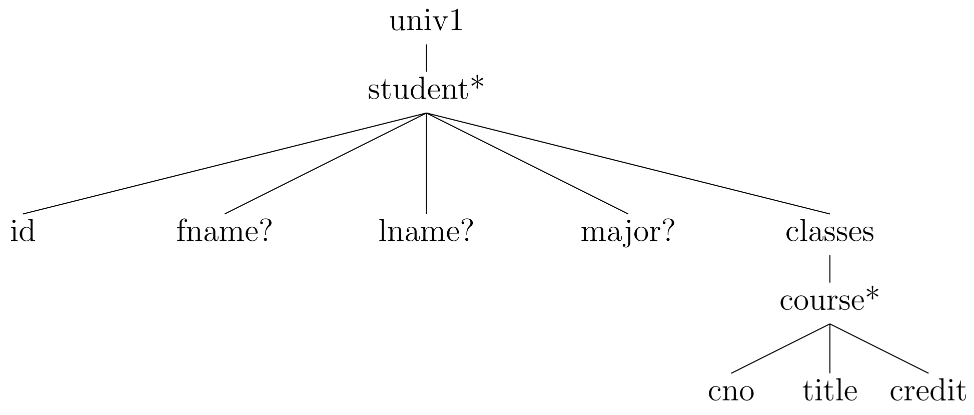

Figure 1.2: Tree representation of University 1

source information to appropriate binary predicates of a specified ontology. Ideally each database administrator of a source involved in transmission/retrieval of data, maps their local XML data to a set of predicates. Previous discussions of the development of tools to assist administrators brought about talk of mapping tools.

The creation of a predicate mapping consist of mapping two nodes and one associating element, called the glue, to a predicate. The glue illustrates a common ancestor (least common ancestor in most cases) of the two nodes of the mapping. The proposal of a graphical interface leads to the possibility of simplifying the mapping process. This interface would present a XML document illustrated in tree form which would allow users to select three nodes appropriate for the arguments of a predicate. Figure 1.1 displays a tree representation of a XML document simulating the proposed graphical interface. If a person was to consider the mapping of information from a source to the predicate `id-name`. The expression to do so would appear is depicted by Figure 1.3.

```
id-name($I, $N) ← univ1/student $G, $G/@id $I, $G/name $N
```

Figure 1.3: A subset of XPath is used to represent mapping in [7] and will be used in this thesis likewise.

Using the graphical interface to accomplish this involves selecting the `@id` and `name` child nodes of students for the first and second predicate arguments respectively and the `student` node as the glue.

With the initial focus of enabling interoperability complete, lets direct attention to the process of query composition and processing. Selection, Join, and Projection are the basis of the queries discussed in this research. The bulk of the query processing revolves around the join of the necessary binary predicates required to obtain the desired results. The expansion of a user query leads to the union of predicate fragments from all sources joined together. Further expansion leads to a set of subqueries consisting of all permutations of joins. An example is provided to clarify the concept of the composition of queries. Consider a query where the results consist of the title of courses taken by students with the data deriving from two sources. Based on the ontologies previously provided this query would appear as $\texttt{id-cno} \bowtie \texttt{cno-title}$. Expansion of the query leads to $(\texttt{id-cno}_1 \cup \texttt{id-cno}_2) \bowtie (\texttt{cno-title}_1 \cup \texttt{cno-title}_2)$. Further expansion of the query leads to the desired expression as it appears in Figure 1.4.

$$\begin{gathered}(\underline{\texttt{id-cno}_1 \bowtie \texttt{cno-title}_1}) \cup (\texttt{id-cno}_1 \bowtie \texttt{cno-title}_2) \cup \\ (\texttt{id-cno}_2 \bowtie \texttt{cno-title}_1) \cup (\underline{\texttt{id-cno}_2 \bowtie \texttt{cno-title}_2})\end{gathered}$$

Figure 1.4: Union of all subqueries.

For n sources and m predicates, there are $\bigcirc(n^m)$ subqueries to possibly evaluate. The final expansion of the example query shows this by the number of fragments unioned. Of those subqueries, only $\bigcirc(n)$ can be executed locally [7]. The n local subqueries exemplify the notation of intra-source processing. The concept refers to when all fragments of the subquery share the same source. The underlined subqueries in Figure 1.4 identify the local subqueries. In the case where there are n^m subqueries, outside of the n local queries, inter-source processing must occur. The evaluation of those subqueries require communication between multiple sources, thus creating additional overhead.

Inter-source processing provides additional overhead to the framework, but [7] provides a method of identifying subqueries requiring no inter-source processing.

Consistency Condition: We say a predicate p satisfies the *consistency condition*, provided: (i) whenever a constraint $p(\alpha) \leftarrow p(\beta)$ holds in p, then it holds for every fragment p_i; (ii) if for fragments p_i and p_j we have $t_i \in p_i, t_j \in p_j$, and $t_(\alpha) = t_j(\alpha)$, then $t_i(\beta) = t_j(\beta)$.

Let Q be a query involving the join of $p(A, B)$ and $q(B, C)$, and Q also has the distinct clause to eliminate duplicates. Let k be a source.

Theorem 1 *No inter-source processing involving p_k is needed if and only if the following conditions hold: (1) The key contraint $B \leftarrow C$, and the consistency condition hold for q, and (2) There is a foreign key contraint from p_k to q_k on the attribute B.*

Theorem 4.1 of [7] formally states when the elimination of inter-source processing can take place. Particular inter-source subqueries can be eliminated when the consistency constraint and a key constraint between the elements in the second binary predicate hold, and there is a foreign key constraint from the first binary predicate (its second variable) to the second binary predicate (its first variable) on an element of both. With regards to the expanded query as shown in the last example, if the consistency constraint [7] holds for cno-title (regardless of the source, a course will always have the same title) and there is a foreign key constraint from id-cno to cno-title on the course attribute, all inter-source queries can be eliminated involving sources where that holds true. Eliminating inter-source processing essentially boost system performance in all cases.

Moreover, the concepts brought about in [7] establish a strong foundation for the development and further advancement of a lightweight infrastructure that successfully enables interoperability. The idea eliminates some of the shortcomings associated with a “global schema” by enriching local source with declarations based on an “application specific ontology“ composed of binary predicates. With the advancement of knowledge presented by [7] and the development of the tools mentioned, the lightweight infrastructure should help make advancements in the area of interoperability and data integration.

CHAPTER II

CONTRIBUTIONS

2.1 Generation of Subqueries

2.1.1 Concept of Subqueries

A major benefit of applying a global schema is that it allows users to pose queries, evaluated over multiple sources, with knowledge of only the global schema. In addition to that, sources need not alter their schemas for the inclusion of data in the results of queries other users (possibly located at alternate sources) request. It is only required that a mapping exist from sources to the global schema. In this research the ontology, in a sense, provides the global schema. Those sources are the ones who desire the inclusion of their data. For both qualities mentioned to hold true simultaneously, translation from a query composed over the semantic model into multiple queries must occur. The execution of each query needs to take place at a source which posses data from all sources involved in the query.

In the framework, the coordinator handles the tasks of managing and translating queries. The coordinator translates a global query into a group of queries

Univ1(student info)

id	name	major
50231	Jack	CSC
50457	Mark	MTH
50825	Wendy	BIO

Univ1(registration)

id	cno	title
50231	656	Foundations of CSC
50457	555	Algo. Analysis
50825	671	Adv. Databases

Table 2.1: A listing of student ids, their names, and course numbers for University 1.

Univ2(student info)

id	name	major
50322	Jane	CSC
50411	Jim	MTH

Univ2(registration)

id	cno	title
50457	705	Computer Vision
50322	692	Adv. Wireless Networking
50411	605	Adv. Operating Sys.
50411	720	Human Comp. Interaction
50231	720	Human Comp. Interaction

Table 2.2: A listing of student ids, their names, and course numbers for University 2.

which not only (1) return the complete results based on each individual source but (2) take into consideration when one source may not harbor all the information required to answer the query completely. Upon gaining information from alternate sources complete answers become obtainable.

Example

Provided is an example to give better understanding of the subqueries the coordinator creates, illustrating a specific instance of the scenario at hand. Tables 2.1 and 2.2 exemplify two separate universities where each university maintains information about its full time students and the course registration of all students. The two universities are integrated in the sense that students can register for courses at the university where their enrollment is less than full time. This thesis makes the assumption that all students are enrolled full time at only one university. Here the university where the student is not enrolled full time is referred to as a *alternate university*. Additionally, students enrolled full time at a university are regarded to as *primary students*. A student is identifiable by a single id, regardless of the university. When a student registers for a class at an alternate university, that school lacks information about the student aside from their registration information (e.g. cno, title). That indicates that there is no foreign key relationship between the student info table and the registration table (at a particular university) on the student id attribute. Suppose one would like to determine the name of every student from Univ1 and Univ2, and the title of all courses in which they are enrolled (at either university). Performing a natural join of the student info table and the registration table at Univ1 on the id attribute, selecting the name and title of each record, and projecting that information unioned with the corresponding information of Univ2 seems as an appropriate approach to the results. Evaluating the query in that fashion leads to the results in Table 2.3.

Table 2.3 corresponding to the results of the query mentioned actually displays incomplete results. The results fail to list courses students are registered for

Results

name	title
Jack	Foundations of CSC
Mark	Algo. Analysis
Wendy	Adv. Databases
Jane	Adv. Wireless Networking
Jim	Adv. Operating Sys.
Jim	Human Comp. Interaction

Table 2.3: Incomplete Query Results

Results

name	title
Jack	Foundations of CSC
Mark	Algo. Analysis
Mark	Computer Vision
Wendy	Adv. Databases
Jane	Adv. Wireless Networking
Jim	Adv. Operating Sys.
Jim	Human Comp. Interaction
Jack	Human Comp. Interaction

Table 2.4: Complete Query Results

at an alternate university. The lack of a tuple indicating that Mark is enrolled in the course titled Computer Vision and that Jack is enrolled in Human Comp. Interaction illustrates this point. The query actually returns the names of all students and the courses they are enrolled in at their primary universities. To obtain the results that is sought requires the union of these results with the results of (1) student info table of Univ1 joined to the registration table of Univ2 on the id attribute and (2) the student info table of Univ2 joined to the registration table of Univ1 on the id attribute. These additions allow for the inclusion of results where a student is registered for classes at an alternate university. This concludes with the complete results as they appear in Table 2.4. In the example above, both University 1 and University 2 share the same schema. That is not an assumption made in the framework, and presents another issue to overcome. Thus, the use of a semantic model resolves issues pertaining to the heterogeneity of sources' schemas.

Classification of Subqueries

[7] introduces an approach which encapsulates both points that the coordinator must take into consideration. That approach consist of generating subqueries to be executed at particular sites (with access to all data sources involved in the subquery). The establishment of these subqueries involve identifying global predicates in the user query and replacing them and other items described in terms of the predicates with the appropriate information. The source mappings furnishes the information necessary to replace the binary predicates in the user query. An example source mapping is located in Appendix F.

Subqueries fall into one of two categories:

1. Local subqueries: a query derived from the global query translated in terms of a single source mapping.

2. Inter-source subqueries: a query derived from the global query translated in terms of multiple sources mappings (involves data of more than one source).

Evaluating local subqueries requires the data of exactly one source. Complete evaluation of these subqueries requires no prior data transmission. On the other hand inter-source subqueries need data from multiple sources to completely evaluate the subquery. Data transmission occurs to ensure the availability of the appropriate data from various sources for the processing of the subquery, and this thesis provides discussion on the notion of data transmission.

2.1.2 “AllSubqueries” Algorithm

With the conceptual knowledge pertaining to subquires explained, the question that must be answered is, how does one arrive at these subqueries? [7] provides an algorithm which given a global query Q and a source mapping M_i for source i outputs a local subquery Q_i at source i resulting from Q. The complete algorithm is located in Appendix A. In this thesis, extension of the algorithm occurs to allow to the generation of inter-source subqueries as well. The acquisition of the desired results require modifications to the input and output accordingly. The input to generate all subqueries requires the global query Q and all n source mappings M. The corresponding output consists of all possible subqueries. In Algorithm 1 n and m respectively correspond to the number of sources and number of predicates involved in the subquery. The array $a[i]$ provides an indication of which source mapping provides the translation information for the i^{th} (unique) predicate in the user query. For example, $a[2] = 5$ indicates that the second predicate in the user query and everything evaluated in terms of a variable assigned to that predicate is translated

Algorithm 1 AllSubqueries algorithm

```
procedure AllSubqueries($Q, M$)
    $n \leftarrow$ number of sources
    $m \leftarrow$ number of predicates involved in global query $Q$
    $a[m] \leftarrow$ array of size $m$
    $x \leftarrow 0$
    for $i \leftarrow 0, n^m - 1$ do
        $a[x] \leftarrow a[x] + 1$
        while $a[x] == n$ do
            $a[x] \leftarrow 0$
            $x \leftarrow x + 1$
            $a[x] \leftarrow a[x] + 1$
            if $x == n$ then
                exit while loop
            else$x \leftarrow 0$
            end if
            for $j \leftarrow 0, m - 1$ do
                if source$a[j]$ does not contain predicate $j + 1$ then
                    $dontStore \leftarrow TRUE$
                    exit for loop
                else
                    $ReplacePredicate(j + 1, a[j])$
                    $dontStore \leftarrow FALSE$
                end if
                if $dontStore == FALSE$ then
                    $StoreSubquery()$
                end if
            end for
        end while
    end for
end procedure
```

in terms of source five. So for a given global query, m replacements specified by [7] algorithm for each subquery occurs. Instead of using the i^{th} source to translate the entire global query at once, now the source specified on the particular iteration of the `for loop` is used to translate each predicate individually. Those actions correspond to the *ReplacePredicate* routine. The routine takes two parameters, (1) the position of the predicate and (2) the source used when translating the predicate. That statement executes m times for each of the n^m iterations of the outer `for loop`. If a global query entailed three sources and four predicates, the sequence of inter-source subqueries produced would consist of, 1-1-1-1, 1-1-1-2, 1-1-1-3, 1-1-2-1, . . . , 4-4-4-4. Following the translation of the global query, the subquery is stored by the coordinator (the *StoreSubquery*() routine) until transmission time. Subqueries where a predicate fragment is not available are excluded using the *dontStore* boolean, and this is because the completely translation of the global query is not possible. As previously mentioned, sources may not provide a mapping to a predicate for various reasons. When they do not there is no need for a subquery which contains that predicate fragment. No useful information can be acquired due to the fact that the partial information possibly obtainable from the subquery will appear in results of other subqueries.

2.1.3 Application of Optimization Techniques

[7] proves that obtaining complete results does not require all subqueries under particular circumstances. The algorithm discussed above disregards that during its generation of subqueries. To overcome this hurdle, the algorithm previously described can be replaced with an alternative algorithm, and this algorithm takes the required subqueries as input. Each subquery contains m predicate-source pairs. The output would consists of all subqueries corresponding to the requirements. The

coordinator generates each subquery using its respective predicate-source pair. This corresponds to the *ReplacePredicate* routine that was previously described. Instead of passing the position of the predicate and the source it uses for translation, the predicate to be replaced and the source used for translation are provided as input.

Another alternative involves creating all the subqueries but distinguishing which queries to transmit when obtaining the results to the query, and this consists of storing the subqueries with additional information distinguishing which sources where involved in each subquery. With this approach, once the coordinator computes which subqueries are needed and where they should be transmitted, the selection and transmission of the required subqueries is possible.

2.2 Efficient Data Transmission

2.2.1 Evaluation of Subqueries

The system is presented a query in terms of the global binary predicates (using relation algebra). The query entered by the user involve a join of two predicates, reading as `id-cno` $\bowtie$ `cno-title`. For a system where there are i sources, each of the i sources possibly contains a fragment of the predicate p, denoted p_i. The global predicate p is equal to the union of all i fragments (without duplicates). For the example presented, `id-cno` is equivalent to $\texttt{id-cno}_1 \cup \texttt{id-cno}_2 \ldots \cup \texttt{id-cno}_i$

and $\texttt{cno-title}$ is equivalent to $\texttt{cno-title}_1 \cup \texttt{cno-title}_2 \ldots \cup \texttt{cno-title}_i$. Expansion of the global query must occur for further evaluation. This is accomplished by replacing the predicates of the query with the union of their fragments. $(\texttt{id-cno}_1 \cup \texttt{id-cno}_2 \ldots \cup \texttt{id-cno}_i) \bowtie (\texttt{cno-title}_1 \cup \texttt{cno-title}_2 \ldots \cup \texttt{cno-title}_i)$ which evaluates to $\texttt{id-cno}_1 \bowtie \texttt{cno-title}_1 \cup \texttt{id-cno}_1 \bowtie \texttt{cno-title}_2 \bowtie \ldots \texttt{id-cno}_i \bowtie \texttt{cno-title}_i$. Processing of the inter-source subqueries requires transmission of the required data to a central location. This situation brings about the issue of how to determined where to send the data.

2.2.2 General Idea

Obtaining the subqueries places the concept one step closer to determining the results of the global query. The coordinator is responsible for handling the processing and distribution of the subqueries. Section 2.1.2 of the thesis presented techniques for the generation of the subqueries. Now the transmission of the subqueries to the correct sources must occur. After transmission, subqueries are at sources which contains the appropriate combination of data from all sources.

To allow the joining of predicate fragments, the sources containing the data fragments must transfer their data to a central location. At the location the evaluation of the subquery may take place. For simplicity, an assumption is made that the cost of transmitting the data is equivalent to the size of the data (XML document) transmitted. Additionally, one has to assume a uniformed network for the data transmissions (i.e the transmission of data from a particular source to any other node is identical). For example, transmitting data from source one to source two and transmitting from source two to source five have identical cost. For a source to acquire the data required for a local subquery involving itself requires zero units.

2.2.3 Processing Subqueries Using the Coordinator

One method of establishing a centralized location for the processing of subqueries involves transmitting the data from all the sources to the coordinator. With this technique, the coordinator may request for the data of all sources. While waiting for sources to complete transmission of data, the simultaneous generation of the subqueries provides good use of potentially idle time. In this method data transmission cost is irrelevant. The coordinator will send all sources involved in the query a data request and the corresponding sources respond by sending their data. This approach solves the issue of having all the data in a centralized location. The question then becomes, is this the most efficient way of handling the situation? The coordinator will now be performing an extraneous amount of work, while all the sources are idle. Another question that arises is whether or not there is a way to limit the sources that actually transmit their data? With these questions, some good points arise and addressing those issues lead to the next approach.

2.2.4 Processing Subqueries Using the Largest Source

Proposed is a data transmission technique where the $n-1$ smallest data sources transmit their data to the largest source. That is where all the subqueries will be evaluated and the results of the global query will be determined. Such an approach relieves the coordinator from having to do the processing of the subqueries by assigning them to a source to process. In doing so, it also reduces the number of data sources which transmit data, thus reducing the amount of data that needs to be transmitted to evaluate the global query. The approach also overcomes the shortcomings identified in the method above. Although this approach appears more sufficient than the previous, the question remains as to whether it is always the best approach. Earlier in the thesis there was discussion of when inter-source queries can be eliminated. If this were so, then it may not be necessary to send all the data to a single source to have it perform all required processing. It is possible that there may be a situation where half of the sources can be processed at one node and the other half at another node; however this is all depending on the inter-source subqueries that are required. For all required subqueries, the set of sources for each one is added to the group of sources that need to all be in a common location. Each group that is a subset of another group can be eliminated and once all subqueries have been evaluated, the set of groups that has been established shows which sources need to share a common location. The next approach uses this information and determines an optimal solution for efficient data transmission which provides the optimal approach under certain conditions. If for all sources i, $j \in \{s_1, s_2, \ldots s_n\}$ there exist a subquery that requires both i and j, then this approach will provide the optimal solution. There may be other situations where this also provides the optimal solution.

2.2.5 Naive Approach to Processing Subqueries

Although the previous method can produce the optimal solution in some cases, it has its share of shortcomings. The naive approach is presented to overcome those shortcomings and provide the optimal solution in every scenario. Here iteration through every possible combination of data transmission between the sources involved occurs. For each iteration (1) check to see if all subqueries are able to be processed with the particular transmission pattern and if so (2) determine if it has less of a cost than any of the other previous iterations. If it cost less, store that solution and the cost and continue to iterate through the possibilities until they all have been exhausted. Now an approach has been presented which will always produce the optimal solution, but its disadvantages are costly. The time complexity of this algorithm is $2^{n(n-1)}$ which is well beyond acceptable for the approach when applied to even a somewhat large number of sources. For the processing of a query which resulted in the need for all local and inter-source subqueries, the most efficient scheme of transmitting the data would be to ship the data from each source to the source which had the largest unit of data to transfer.

2.2.6 Manipulation of Properties of the Framework

The naive approach is a very costly option, but always results in an optimal solution. The algorithm presented does not always present an optimal solution, but is more efficient.

Source localization combinations

The number of possible subqueries is of $\bigcirc(n^m)$, and the transmission of data required to evaluate particular subqueries may overlap. There is no need to use the list of required subqueries because it is extremely likely that the list can drastically be reduced to a form that brings efficiency to obtaining an optimal transmission scheme. Two subqueries involving the same combination of sources may produce different results because each source may be applied to a different predicate. The subqueries `id-cno`$_1 \bowtie$ `cno-title`$_2$ and `id-cno`$_2 \bowtie$ `cno-title`$_1$ are completely different subqueries. But for data transmission only the combinations of sources is relevant. In combinations, as in set theory, order has no relevance. In the previous example sources one and two were used in the first subquery and sources two and one were used in the second. Both subqueries require the same data localization of data because the two sets of sources required are the same.

To obtain the simplified source combinations the evaluation of each subquery as a set must occur. The sets are reduced such that no set is a subset of another set included. The subqueries are treated as sets not multisets. The subquery `id-cno`$_1 \bowtie$ `cno-title`$_1$ is evaluated as the set $\{1\}$. Now as opposed to having at most n^m situations to take into consideration there are at most $n(n-1)/2$ when $n > 1$ possibilities to evaluate.

Obtaining "Optimal Solution"

To determine the optimal solution requires the use all possible pairs (with the criteria above being met) of size two. The pairs are determined by first obtaining a list of the sources ordered by size and from this list the pairs are obtained. This list contains all combinations of sources. The pairs are obtained from the list as follows. Evaluation of a pair occurs only if a combination appears in a subquery.

Then data transmission from the source to the destination of the two sources of the pair takes place if the two do not already preside at a common node. This results in an optimal solution some of the time. Usually this will produce optimal solution if no two queries overlap on more than one source. It is possible for this algorithm to produce the optimal result when overlapping occurs, but there are certain criteria that must be met for this to occur. Further research is required to obtain an algorithm that obtains the optimal solution for data transmission in all cases.

Merger of Source Combinations

In certain situations merging combinations of sources prior to evaluations with the algorithm described above allows it to produce an optimal solution. If data overlaps between two combinations and only two combinations, then if for one of the two combinations the cost of transmitting the overlapping data is more than transmitting the remaining sources then the two combination sets can be unioned and evaluated as one set. Upon applying the algorithm on the subqueries, an optimal solution will be obtained.

2.3 Obtaining Minimal Intermediate Data (Relational)

The concept of inter-source subqueries demands the transmission of data from various sources to allow predicate fragments required in a particular subquery to reside in a common location. The ability to minimize this aspect of the system potentially reduces an enormous amount of system overhead. This section provides a new approach attempting to accomplish just that. One of the major benefits of XML is its flexibility. With this technique, its flexibility provides us with multiple cases to take into consideration. During research three situations regarding the XML's schema have been identified which includes:

1. A top down hierarchical model where a relationship between an attribute is identified only by its predecessors.

2. "Key/Foreign Key" approach where an identifier is used to relate an element in one subtree to data in another subtree where the data elements are not siblings. This corresponding to id-idref and other similar approaches to an XML schema.

3. A schema using id-idrefs.

The primary focus of this thesis is on the first case listed above. There is some discussion about the second situation. Case three is not mentioned in more details.

Currently no mapping techniques have been implemented to handle the situation.

2.3.1 Relational Approach

Here lies a proposal of a alternate method to the prior technique of using local and inter-source subqueries. The approach discussed previously consist of individual sources transmitting the data among other sources in a fashion that allowed all the processing of subqueries. The new approach involves gathering all the information from each source possibly useful. Each source is queried to obtain its complete data (which provides the results for the local subquery) and any incomplete data, possibly useful in performing the necessary joins to other sources.

These intermediate results are presented in *relational* form. Each tuple of the results contains either an attribute exactly once, or a null value for the attribute or the absence of elements. Null values appear when a source does not contain all the information required to present a complete tuple. One can relate this technique to a *full outer join.* With a full outer join of two predicates, when the attribute joining the predicates does not correspond to an attribute in the tuples of the opposing predicate, null values appear in the columns of the predicate lacking data. This can be expressed in relational algebra using its basic operators on binary predicates r

and s as

$$(r \bowtie s) \cup ((r - \Pi_R(r \bowtie s)) \times \{(null)(null)\}) \cup ((s - \Pi_S(r \bowtie s)) \times \{(null)(null)\}) \quad (2.1)$$

2.3.2 Key Concepts

With the use of XML and XQuery, sources' schemas do not have to follow suit with the output of the final results. Another benefit of using XML is that it should have a top down hierarchy (for case one) and one root element which every other node in the tree is an ancestor of. From this, the assumption is made that every element and its ancestors possess some relationship. With that being said, note that every group of predicates can be written in terms of some least common ancestor (with the root being the least common ancestor of every element in the tree).

Sources provide mapping rules for each of the binary predicates it contains data for. These mappings describe the predicate in terms of the particular source. As previously stated the glue ($G) variable is the least common ancestor of the two attributes of the binary predicate. The mappings identify the glue and describes both attributes in terms of the glue variable. Figures 2.1, 2.2 and 2.3 display mappings for predicates `id-name`, `id-cno` and `cno-title`. For the predicate `id-cno`, the least common ancestor of the two attributes is `/univ/student`. This indicates that for each student in the university, the predicate `id-cno` list all combinations

```
id-name($I, $N) ⟸  univ/student  $G,
                   $G/id         $I,
                   $G/name       $N
```

Figure 2.1: Mapping for id-name.

of their `ids` and `cnos`. In this case `id` describes exactly one student, so for each `id` it would list all `cnos` that correspond to the student.

```
id-cno($I, $C) ⟸  univ/student            $G,
                  $G/id                   $I,
                  $G/classes/course/cno   $C
```

Figure 2.2: Mapping for id-cno.

```
cno-title($C, $T) ⟸  univ/student/classes/courses  $G,
                     $G/cno                        $C,
                     $G/title                      $T
```

Figure 2.3: Mapping for cno-title.

Building off of the mapping rules for each predicate, this thesis derives a mapping rule for the joining of predicates in a user query. Using the same notation used in [7] the query `id-name` ⋈ `id-cno` ⋈ `cno-title` results would appear as Figure 2.4. The Figure 2.4 depicts each attribute in terms of one of its ancestors. To obtain this first involves determining the glue of all predicates included in the join. Upon accomplishing that the glues are written in terms of the closest ancestor which happens to also be a glue.

```
id-name-cno-title($I, $N, $C, $T) ⟸  univ/student  $G1,
                                     $G1/course    $G2,
                                     $G1/id        $I,
                                     $G1/name      $N,
                                     $G2/cno       $C,
                                     $G2/title     $T
```

Figure 2.4: Mapping for query results.

"GenerateGlues" Algorithm

An algorithm has been derived to create the mapping for the joining of predicates. The input consist of a list containing each predicates glue and a list of the complete paths of the predicate attributes. The algorithm outputs the mappings for the joining of all predicates in the input.

The glues are sorted lexicographically (reading the path from left to right) in ascending order. After successfully sorting the glues, the FindGlue routine determines the least common ancestor of all items in glueList. This, in most cases, will be the first glue in the newly sorted list. The joinMappings array stores sets, with each set containing (1) the full path of the glue, (2) the glue in terms of a variable (exception being the least common ancestor) and (3) the new variable for which the glue used to describe the path. For the least common ancestor of all glues the path for the first two arguments and a new variable declaration are stored. After determining the super glue the remaining glues are processed. Iteration through the glueList starting from position one of the array follows. If by chance, the least common ancestor of the glues was not included in the list position zero becomes the starting position. The variable tPath is used to temporarily store the glue of the current iteration. The next loop entered iterates downward starting from the number of levels in the path of tPath, counting down to one. Another temporary variable tmp is used to store the path of tPath up to the j^{th} element. The variable tmp is compared against all glues currently in the glueMapping variable. If a glue corresponding to tmp does not exists tmp is evaluated with the last element removed from the end of the path expression. If tmp is found in the list of joinMappings the next step involves ensuring that a glue equal to tPath does not already exist. If that is also true addition of the set containing tPath to joinMappings occurs, the path derived from replacing the segment equal to tmp in tPath with the variable that corresponds to the glue tmp, and a new variable declaration. After adding the new set to joinMappings or discovering one already exists the inner `for loop` is exitted and continue processing the rest of the paths in the glueList. Once all glues in glueList have been evaluated the evaluation of the attributes of the predicates must take place. The process is almost identical to that of processing the glues with a couple of exceptions. The number of levels evaluated in the inner `for loop` is one less because it is known that it is an attribute. The other difference is that instead of storing a glue for the first argument nothing is stored because it is not a glue and nothing will be expressed in terms of the attribute.

The example of the mapping in the Figure 2.4 presents a mapping for the join `id-name` $\bowtie$ `id-cno` $\bowtie$ `cno-title`. When applying the algorithm to this problem, the input includes

- `/univ/student`
- `/univ/student`
- `/univ/student/classes/courses`

for the glueList and

- `/univ/student/id`
- `/univ/student/name`
- `/univ/student/classes/courses/cno`
- `/univ/student/classes/courses/title`

for the attribute list. The glues are sorted in lexicographical order. This results in the list as follows

- `/univ/student`

- `/univ/student`
- `/univ/student/classes/courses`

Next determining the least common ancestor to all of the glues takes place. The super glue corresponds to the first glue in the list which is `/univ/student`. That glue is stored in joinMappings along with the glue again and a variable, $G1 for this example.

Evaluation of the outer `for loop` follows. The assignment of glueList[1], which contains `/univ/student`, to tPath occurs next. With the variable j equal to two a check occurs to see if the glue `/univ/student` is contained in the joinMappings, which it is. Subsequently a check occurs to see if the tPath already exists in the joinMappings. In this case and it does, so the `for loop` is exited. On the next iteration evaluation of the glue `/univ/student/classes/courses` takes place. This starts with the assignment of the path to tPath. With the full path assigned to tmp, a check occurs to see if joinMappings contains tmp, which is false. Next /univ/student/classes is assigned to tmp and once again it is not in joinMappings. Following that `/univ/student` is assigned to tmp and it is contained in joinMappings. The algorithm then checks if tPath already exist in joinMappings which is false which leads to it being added. The first argument when adding to joinMappings will be tPath which is `/univ/student/classes/courses`. That is followed by replacing tmp in tPath with the variable declaration of tmp. The algorithm replaces `/univ/student` in tPath with \$G1 resulting in \$G1/classes/courses. The last argument of the set to be added to joinMappings is a new variable which is denoted as \$G2.

With all the glues processed, the algorithm processes the predicates. Starting with the assignment of `/univ/student/id` to the variable tPath. The inner `for loop` then iterates downward starting at the level count of tPath minus one. This is equal to two in this case. Comparisons are made to determine if `/univ/student` is contained in joinMappings and it is. The absence of the attribute in joinMappings leads to the insertion of the set consisting of NULL for the glue (because attributes are being processed) along with \$G1/id and \$I for the new variable. This same process is carried out for the remaining predicate attributes resulting in the information necessary to formulate the glue mappings depicted in the Figure 2.4.

2.3.3 Obtaining the XQuery

The proposed framework requires the querying of XML data using XQuery. The use of XQuery provides a way to obtain the data in the form described above. Although XQuery has grown to become a powerful language, it lacks a straight forward method to present results in the form desired. This is because XQuery does not directly support full outer join operation. When performing a join in XQuery, it corresponds to a natural join if attributes are specified for the join or a cartesian product if none are specified.

2.3.4 Overview of Approach

The concepts presented in this section form an approach that differs from the method involving subqueries presented in [7]. Presenting the results of each source in a relation form presents various benefits. Although concepts surrounding the idea of the relational approach must be extended, a far comparison can still be made. This approach provides complete and incomplete tuples from every source. Next would be to determine how to process these results and obtain the complete answers to the global query.

2.3.5 Merging Results

With the prior approach of obtaining the result of the global query, subqueries were used. Each subquery consisted of results that were to appear in the final results. Once all the subqueries were executed, the result of the global query was determined by unioning all of the individual subquery results. The approach discussed in this section eliminates the use of subqueries. Instead there is one query issued at each source. And instead of the query consisting only of results to appear in the results of the global query, they now consists of results of a single source to appear in a subquery, along with partial information which possibly may not be used at all. To obtain the result of the global query involves more than a union.

The approach first involves unioning the data from all the sources. Next a primary key of the data in relational form must be identified. For each item of the primary key the distinct values corresponding to it and the distinct values of each attribute that the item functionally determines are to be retrieved. This approach only returns tuples which include every attribute required by the global query. The complete results are now available.

2.3.6 Current Limitations

As mentioned three possible scenarios of XML schemas have been identified. Of those three an algorithm is only presented for one, which is the top down hierarchy. Type two is described and mappings are discussed, but no algorithm is presented to obtain the XQuery necessary to produce results from the source which data is formatted accordingly. With type three there is currently no mapping specifications. No efforts were made to consider this type for relational form. There have been no optimization efforts made pertaining to this new technique. There is room for optimization and further research efforts should identify and provide implementation for optimization.

2.4 Compare Approaches

The two approaches attempt to accomplish the same goals but take two different paths in obtaining the desired results.

2.4.1 Number of Queries to Execute

The first approach requires at most n^m subqueries. This is because it must join each predicate fragment of a source to every other combination of predicate fragments. With the relational technique only one query is required for each source. This results in $n^m - 1$ less queries to be executed.

2.4.2 Quantity of Data Retrieved by Subqueries

With the subquery approach every result of the subquery will appear in the results of the global query. Each subquery does not return any extraneous data. However there is the possibility that some results may repeat across subqueries though. With the relational approach there is extraneous data. The queries executed at each source returns the complete results for that particular source as well as other information that might be useful with information from other sources. The problem is that other sources may not need the extra information. With that being said, this research now encounters the fact that this method may return far more data than what is actually needed to obtain the query results. There is the possibility that some of the extraneous data can be eliminated with the where clause of the global query when obtaining the intermediate results.

2.4.3 Optimization Techniques

No optimization techniques have been identified corresponding to the relational technique. On the other hand techniques to optimize the subquery approach exist. Optimization efforts include minimizing the number of subqueries to execute. Even with this technique the least amount of possible subqueries would still equal the number of subqueries required in the relational approach.

2.4.4 Data Transmission

Both approaches require some sort of data transmission. With the subquery approach corresponding to methods presented, the data of each subquery must reside in a common location. This requires the transmission of the documents. With the relational approach data transmission is only required to send the intermediate results back to the coordinator where they are unioned together to form the final results.

2.4.5 Distribution of Workload

With the relational approach each source is assigned one subquery to execute and then the coordinator executes one query to union all the results. With the subquery approach the execution of queries can in some situations be distributed about the sources and the results can even be unioned at various sources prior to transmission to the coordinator.

Algorithm 2 GenerateGlues algorithm

```
1: procedure GenerateGlues($G[], P[]$)    ▷ List of glues and list of predicate attributes (full paths included)
2:     $glueList[] \leftarrow G[]$    ▷ Glues for all predicates involved in join
3:     $predList \leftarrow P[]$    ▷ Paths of predicate attributes where predicates involved in join
4:     $Sort(glueList[])$
5:     $glue = FindGlue(glueList[])$
6:     $joinMappings[] \leftarrow glue, glue, newvariable$    ▷ new variable i.e G0
7:     for $i \leftarrow 1,$ number of glues in glueList do
8:         $tPath \leftarrow glueList[i]$
9:         for $j \leftarrow$ levels of tPath$, 1$ do
10:            $tmp \leftarrow$ path of tPath up to $j$ element
11:            if $joinMappings$ contains $tmp$ then
12:                if $joinMappings$ does not contain $tPath$ then
13:                    $joinMappings \leftarrow tPath, Replace(tPath, tmp,$ variable for tmp$), newvariable$
14:                end if
15:            end if
16:            exit for loop
17:        end for
18:    end for
19:    for $i \leftarrow 1,$ number of paths in $predList$ do
20:        $tPath \leftarrow predList[i]$
21:        for $j \leftarrow$ levels of $tPath - 1$ to $1$ do
22:            $tmp \leftarrow$ path of tPath up to $j$ element
23:            if $joinMappings$ contains $tmp$ then
24:                if $joinMappings$ does not contain $tPath$ then
25:                    $joinMappings \leftarrow NULL, Replace(tPath, tmp,$ variable for tmp$), newvariable$
26:                end if
27:            end if
28:            exit for loop
29:        end for
30:    end for
31:    return $joinMappings$    ▷ Output mapping for join
32: end procedure
```

Algorithm 3 RelationalXQuery algorithm

```
 1: procedure RelationalXQuery($Q, M$)
 2:     $predList \leftarrow LoadPredicates(Q)$
 3:     for $i \leftarrow 0,$ Count of predicates in predList do
 4:         $arg1 \leftarrow getArg(M_i, predList[i], first)$
 5:         $arg2 \leftarrow getArg(M_i, predList[i], second)$
 6:         $glue \leftarrow getPathGlue(M_i, predList[i])$
 7:         if $arg1 \neq null$ and $arg2 \neq null$ then
 8:             $insert arg1 into argList$
 9:             $insert arg2 into argList$
10:             $insert glue into glues$
11:         end if
12:         $arglist \leftarrow SortPaths(argList)$
13:         $glueList \leftarrow CreateGlueList(argList)$
14:         $tuple \leftarrow false$
15:         $tmp \leftarrow glueList$ set that contains $argList[0]$
16:         $xquery \leftarrow$ "For " $+$ tmp's variable item $+$ " in " $+$
    tmp's path in terms of variable
17:         for $i \leftarrow 1,$ Count of elements in $argList$ do
18:             if $glueList$ contains $argList[i]$ then
19:                 $tmp \leftarrow glueList$ set that contains $argList[i]$
20:                 if $glues$ contains $argList[i]$ then
21:                     $xquery \leftarrow xquery +$ "let $NULL := "
22:                     $xquery \leftarrow xquery +$ "For $NULL in " $+$
    $NotExistFormat($tmp's path in terms of variable$)$
23:                     $xquery \leftarrow xquery +$ "return <null> "
24:                     $xquery \leftarrow xquery +$ "For " $+$ tmp's variable item $+$
    " in " $+$ tmp's path in terms of variable $+$ " union $NULL"
25:                     $tuple \leftarrow false$
26:                 end if
27:             else
28:                 if $tuple = false$ then
29:                     $xquery \leftarrow$
    $xquery +$ "return <tuple>{ " $+$ tmp's path in terms of variable
30:                 else
31:                     $xquery \leftarrow xquery +$ tmp's path in terms of variable
32:                 end if
33:                 $tuple \leftarrow true$
34:             end if
35:             $xquery \leftarrow xquery +$ "}</tuple>"
36:         end for
37:     end for
38: end procedure
```

CHAPTER III

RELATED WORKS

3.1 PIAZZA

The problem of efficient data exchange and data integration is a long standing problem within the database community. The topics discussed by Halevy et al present a new approach, in an attempt to solve this problem. This concept in a sense tries to combine concepts from networking P2P systems, the mathematical concept of transitivity and data exchange to form a solution to problems apparent in data integration in general and more specifically problems associated with the mediated schema approach.

The brilliance of databases and data management is apparent when looking at how they provide semantically rich data representations and expressive query languages. There are aspects where they have suffered throughout there existence. Two significant problems associated with information exchange in data management and data integration tools include:

- Systems typically requires a comprehensive schema design before they can be used to store or share data.

- Systems are difficult to extend because schema evolution is heavyweight and may break backward compatibility.

To avoid these obstacles some users opt to use applications that provide simpler usability for there everyday task, such as spreadsheets, text files, etc. Although that "solution" accomplishes their immediate task, they lack the advance functionality that is available when using database systems. Varies techniques have been proposed, and some implemented, to alleviate issues associated with large-scale data sharing. Most approaches share the common general idea of using a mediated schema. Mediated schemas provide strong semantics and present data sources with the ability to evolve independent of each other. Although this does provide a solution to the problem, the integrated part of the mediated schema can become a bottleneck. This is clear when you think about the approach and the following aspects associated with it.

- Schema design must be done carefully and globally.
- Significant changes to data sources may cause them to violate the mappings to the mediated schema.
- Only the central administrator has the ability to add new concepts to the

mediated schema.

- Lacks ad-hoc extensibility, which results in limited support for large scale data sharing.

The notion of a PDMS attempts to overcome some of the shorting-comings of prior and some current research efforts geared towards large scale data integration. The goal of the project is to provide a system which will preserve semantics and rich query language, but which facilitates ad-hoc, decentralized sharing, and administration of semantic relationships. Key concepts:

- Implementation of the system will be based on a peer-to-peer architecture, hence the name *peer data management system* (PDMS).
- The system will blend the extensibility of the HTML Web with the semantics of data management tools.

A PDMS can be informally described as a system which consists of a set of data sources, also known as peers, which are related through *semantic mappings*. The assumption is made that each peer defines its own relational *peer schema* whose relations are called *peer relations*. Queries (select-project-join) in a PDMS will be posed over the relations from a specific peer schema. Data can be contributed to the system in the form of stored relations, which can be thought of as data sources. The stored relations are provided by the peers, but every peer need not provide stored relations, they may be in the system to serve as a logical mediator to other peers. The queries posed over the PDMS eventually are evaluated in terms of these stored relations, which can be stored locally or at other peers. PDMS system provides a means of supporting evolving schemas based on attributes that have previously been mentioned. In the case where a schema has evolved, a peer could simply create a new peer relation and map it to the previous peer relation. Such a mapping should be quite simple, mainly because to two relations probably will be closely related.

A Piazza PDMS is composed of a set of nodes (representing physical peers), which are connectors in an overlay network on the Internet. Query reformulation is performed at the node that received the query. The reformulator assumes a global system catalog that provides access to all of the mappings that involve a particular peer.

PDMS provides an easily extensible, decentralized environment for sharing data with rich semantics. It attempts to replace the single logical schema with an inter-linked collection of semantic mappings between peers' individual schemas. And it also provides the ability to exploit transitive relationships among peers, which is key. Lastly, at the moment one aspect that separates it from other notions that try to accomplish similar task is the fact that it overcomes some of the bottlenecks that are apparent in a *mediated schema* approach to data integration.

Both this thesis and [3] attempt to alleviate some common issues associated with data integration. The PIAZZA project attempts to eliminate the bottlenecks that are apparent in a mediated schema approach by using its mapping techniques, while in this research tools brought about by the semantic web are used to accomplish the same goal. Unlike the PDMS, the framework presented in this research will not require users to disregard data they posses. With the PDMS if a source is unable to map attributes to other sources those attributes can not be included in the system. This research eliminates that problem by providing an application specific standard ontology which should consist of "all" attributes from a particular domain.

CHAPTER IV

FUTURE WORKS AND CONCLUSION

4.1 Future Works

The research efforts of this thesis managed to extend the foundation established by [7]. Even with the efforts made, there still remains areas for future research. The knowledge presented in Section 2.2 provides some enlightenment on the issue of data transmission but does not present an efficient technique to always acquire the optimal solution. The naive approach produces the optimal solution in all situations but is burdened by its $\bigcirc(2^{n(n-1)})$ worst case runtime. There were techniques introduced to provide efficiency, but they did not always produce the optimal solution.

Throughout this research the capacity of the various sources involved were disregarded. Currently, it is possible for a data transmission scheme to be used where all documents are to be sent to a single source to process the subqueries. How well can this approach scale? The answer to this question may require further research efforts. This research will need to pertain to how to partially evaluate subqueries with particular fragments across multiple sources and return all result fragments to the coordinator where it can be further processed, and the final results obtained.

In this thesis cost is assumed to be the size of the entire document only, and it was assumed that the network of sources was uniform. Also, the concept of efficient data transmission can be extended to consider the transmission speed between sources as well. Additonally, efficient data transmission can consider cases where direction of transmission differs in speed (i.e. transmission from $s_1 \rightarrow s_2$ differs from $s_2 \rightarrow s_1$). These facts would provide a more accurate picture of transmission cost and the optimal transmission route.

Furthermore, research can also be done in the area of load balancing which would entail the workload of the sources. With the approach presented in Section 2.2.6, it is not uncommon to have sources waiting idle. It may be possible to devise an approach that not only considered cost when transmitting data but also workload. That is to say that even though it may cost more to have two sources containing the same data, it may be faster and provide better overall system performance by having the two sources process subqueries paralleled. This issue could prove beneficial especially when considering applying this to large scale data integration.

Currently, there remain areas of this thesis and the framework established by [7] which lack implementation. Through the current research efforts implementations for the algorithms in Sections 2.1.2 and 2.3.2 were developed. The implementation of the algorithm from Sections 2.1.2 is based off Dongfeng Chen's implementation of the algorithm that appears in Appendix A. The findings which appear in Section 2.2.6 currently have no implementation. Additionally, there remains the task of integrating the algorithms mentioned above with the implementation developed by [8] to illustrate how the framework as a whole would work.

The new approach presented in this thesis is open for further extension. Currently, acquiring the final result of the user query is an area that can be optimized. New ideas pertaining to how to distribute the task of obtaining the final answer among the source instead of assigning the entire task to the coordinator can be established. There is also the possibility of applying optimization techniques previously applied to the subquery approach, and this could possibly reduce the amount of data to transmit.

4.2 Conclusion

Through this research extension of the algorithm for determining local subqueries into an algorithm which produced all subqueries transpired. This research also provided insight to the concept of efficient data transmission. Although there was no discovery of an efficient algorithm to obtain the optimal solution in all cases, a foundation has been laid upon which other researchers can expand. Lastly, this thesis has provided an alternate approach which eliminates the need of inter-source subqueries. With the application of mapping techniques to all predicates in a query, this research is able to obtain complete and partial results from a source which will be merged together with the corresponding data of the remaining sources to obtain a complete answer. With this research some valuable ideas have come about which can be appended to the framework established by [7].

REFERENCES

[1] Sudarshan S. Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The tsimmis project: Integration of heterogeneous information sources. In *IPSJ*, pages 7–18, 1994.

[2] Alon Y. Halevy, Oren Etzioni, AnHai Doan, Zachary G. Ives, Jayant Madhavan, Luke McDowell, and Igor Tatarinov. Crossing the structure chasm. In *CIDR*, 2003.

[3] Alon Y. Halevy, Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu, and Igor Tatarinov. The piazza peer data management system. *IEEE Trans. Knowl. Data Eng.*, 16(7):787–798, 2004.

[4] Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. Data integration: The teenage years. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, *VLDB*, pages 9–16. ACM, 2006.

[5] Zachary G. Ives, Nitin Khandelwal, Aneesh Kapur, and Murat Cakir. Orchestra: Rapid, collaborative sharing of dynamic data. In *CIDR*, pages 107–118, 2005.

[6] Laks V. S. Lakshmanan and Fereidoon Sadri. Interoperability on xml data. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International*

Semantic Web Conference, volume 2870 of *Lecture Notes in Computer Science*, pages 146–163. Springer, 2003.

[7] Laks V. S. Lakshmanan and Fereidoon Sadri. Xml interoperability. In Vassilis Christophides and Juliana Freire, editors, *WebDB*, pages 19–24, 2003.

[8] Jason S. Leone. A distributed approach to xml interoperability. Master's thesis, University of North Carolina at Greensboro, 2007.

[9] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB*, pages 251–262. Morgan Kaufmann, 1996.

[10] Jeffrey D. Ullman. Information integration using logical views. *Theor. Comput. Sci.*, 239(2):189–210, 2000.

[11] www.w3c.org.

APPENDIX A

LOCAL SUB-QUERY GENERATION

Algorithm 1 (Local sub-query generation for an XML source)
Input: Global query Q, source mappings m_i for source i
Output: Local sub-query Q_i at source i resulting from Q.
Method: The idea is to replace variable declarations of Q with variable declarations of source i using the mapping rules m_i. Certain details should be observed in this translation as follows:

1. If a variable declaration $\$X$ in Q corresponds to a *tuple* of a predicate p, then replace it by the declaration for the glue variable in the mapping rule for p in m_i.

2. If a variable declaration $\$X$ in Q corresponds to an argument of a predicate p, and the declaration also has a selection condition on the other argument, then replace it by the declaration for the glue variable in the mapping rule for p in m_i, and include the selection condition as well.

3. If a variable declaration $\$X$ in Q corresponds to an argument of a predicate p, and rule (2) above does not apply, then replace it by the declaration for the argument obtained from the mapping rule for p in m_i.

4. If a URI is used in Q, then Q_i will use its corresponding object id. (Note: we are assuming global queries do not ask for URIs. Otherwise, only for the output, Q_i applies the URI-generating function to the corresponding object id.)

5. Variables declared in Q with respect to an object-objectId predicate do not need a counterpart declaration in Q_i if the object-objectId predicate is joined with another predicate p having the same object as one of its arguments. In such cases the declaration in Q_i corresponding to p can supply the required variable.

APPENDIX B

SCHEMAS

```
<!ELEMENT univ1 (student*)>
<!ELEMENT student (id, fname?, lname?, major?, classes)>
<!ELEMENT classes (course*)>
<!ELEMENT course (cno, title, credit)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT lname (#PCDATA)>
<!ELEMENT major (#PCDATA)>
<!ELEMENT cno (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT credit (#PCDATA)>
```

Figure B.1: DTD: University 1

```
<!ELEMENT univ2 (course*)>
<!ELEMENT course (cno, title, credit, enrolment)>
<!ELEMENT enrolment (student*)>
<!ELEMENT student (id, name?, major?)>
<!ELEMENT cno (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT credit (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT major (#PCDATA)>
```

Figure B.2: DTD: University 2

```
<!ELEMENT univ3 (courses, students)>
<!ELEMENT courses (course*)>
<!ELEMENT students (student*)>
<!ELEMENT course (cno, title, credit, enrolment)>
<!ELEMENT enrolment (id*)>
<!ELEMENT student (id, name?, major?, classes)>
<!ELEMENT classes (cno*)>
<!ELEMENT cno (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT credit (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT major (#PCDATA)>
```

Figure B.3: DTD: University 3

APPENDIX C

TREE REPRESENTATIONS

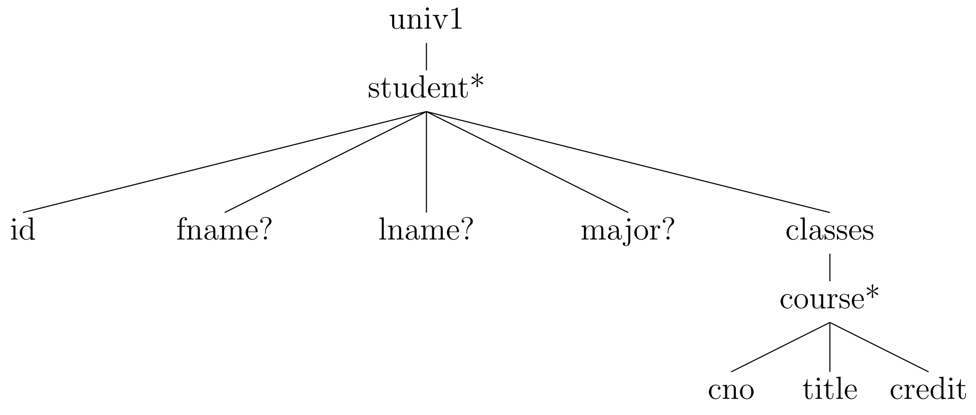

Figure C.1: Tree representation of University 1

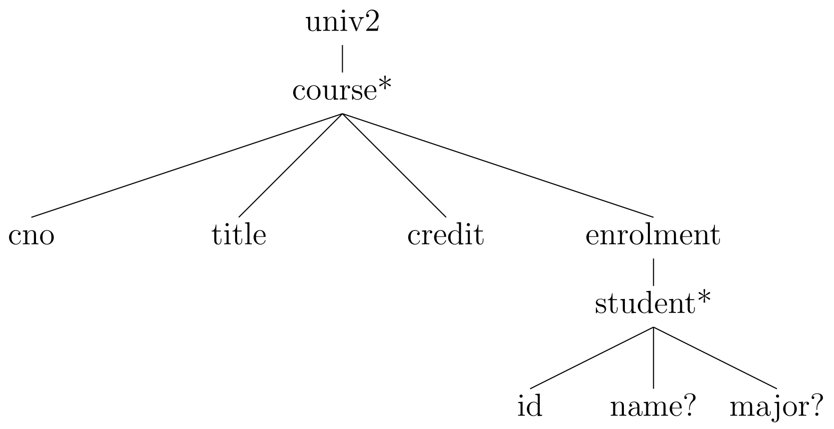

Figure C.2: Tree representation of University 2

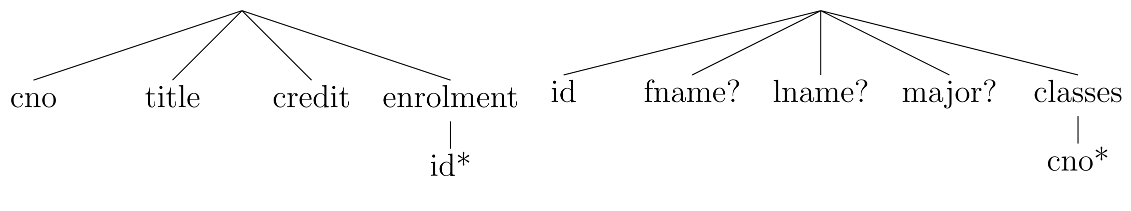

Figure C.3: Tree representation of University 3

APPENDIX D

SAMPLE XQUERY

D.1 Relational Approach

```
for $G1 in doc("univ1.xml")/univ1/student
let $NULL :=
      for $T in $G1[not(exists(id))]
            return <null/>
for $I in $G1/id union $NULL

for $G in $G1/classes/course
let $NULL :=
      for $T in $G1[not(exists($G/cno))]
            return <null/>
for $C in $G/cno union $NULL
let $NULL :=
      for $T in $G1[not(exists($G/title))]
            return <null/>
for $T in $G/title union $NULL

return
      <tuple>{
            $I, $C, $T
      }</tuple>
```

D.2 Merger of Results

```
let $S1 := doc("relational1.xml")/source1/tuple
let $S2 := doc("relational2.xml")/source2/tuple

let $R := $S1 union $S2

for $I in distinct-values($R/id),
       $C in distinct-values($R[id = $I]/cno),
       $N in distinct-values($R[id = $I]/name),
       $T in distinct-values($R[cno= $C]/title)

return
      <results>
            <id>$I</id>
            <name>$N</name>
            <cno>$C</cno>
            <title>$T</title>
      </results>
```

APPENDIX E

SUBQUERY OPTIMIZATION TECHNIQUES

E.1 Approach 1

```
<query>
      <subquery>
            <predicate>
                  <predicateName>id-cno</predicateName>
                  <predicatePos>1</predicatePos>
                  <predicateSource>1</predicateSource>
            </predicate>
            <predicate>
                  <predicateName>cno-title</predicateName>
                  <predicatePos>2</predicatePos>
                  <predicateSource>1</predicateSource>
            </predicate>
                  ⋮
      <subquery>
            ⋮
<query>
```

E.2 Approach 2

```
<query>
      <subquery>
            <fragment>
                  <source>1</source>
                  <predicate>id-cno</predicate>
            </fragment>
                  ⋮
            <filename>subquery1.xq</filename>
      </subquery>
            ⋮
</query>
```

APPENDIX F

SAMPLE MAPPINGS

F.1 University 1

```
<root>
      <mapping>
            <predicate>id-idId</predicate>
            <type>1</type>
            <pathGlue>/univ1/student/id</pathGlue>
      </mapping>
      <mapping>
            <predicate>id-name</predicate>
            <type>0</type>
            <pathGlue>/univ1/student</pathGlue>
            <firstArg>id</firstArg>
            <secondArg>name</secondArg>
      </mapping>
      <mapping>
            <predicate>id-major</predicate>
            <type>0</type>
            <pathGlue>/univ1/student</pathGlue>
            <firstArg>id</firstArg>
            <secondArg>major</secondArg>
      </mapping>
      <mapping>
            <predicate>id-cno</predicate>
            <type>0</type>
            <pathGlue>/univ1/student</pathGlue>
            <firstArg>id</firstArg>
            <secondArg>classes/course/cno</secondArg>
      </mapping>
      <mapping>
            <predicate>cno-cnoId</predicate>
            <type>1</type>
            <pathGlue>/univ1/student/classes/course/cno</pathGlue>
      </mapping>
      <mapping>
            <predicate>cno-title</predicate>
            <type>0</type>
            <pathGlue>/univ1/student/classes/course</pathGlue>
            <firstArg>cno</firstArg>
            <secondArg>title</secondArg>
      </mapping>
      <mapping>
            <predicate>cno-credit</predicate>
            <type>0</type>
            <pathGlue>/univ1/student/classes/course</pathGlue>
            <firstArg>cno</firstArg>
            <secondArg>credit</secondArg>
      </mapping>
<root>
```

F.2 University 2

```
<root>
      <mapping>
            <predicate>id-idId</predicate>
            <type>1</type>
            <pathGlue>/univ2/course/enrollment/student/id</pathGlue>
      </mapping>
      <mapping>
            <predicate>id-name</predicate>
            <type>0</type>
            <pathGlue>/univ2/course/enrollment/student</pathGlue>
            <firstArg>id</firstArg>
            <secondArg>name</secondArg>
      </mapping>
      <mapping>
            <predicate>id-major</predicate>
            <type>0</type>
            <pathGlue>/univ2/course/enrollment/student</pathGlue>
            <firstArg>id</firstArg>
            <secondArg>major</secondArg>
      </mapping>
      <mapping>
            <predicate>id-cno</predicate>
            <type>0</type>
            <pathGlue>/univ2/course</pathGlue>
            <firstArg>enrollment/student/id</firstArg>
            <secondArg>cno</secondArg>
      </mapping>
      <mapping>
            <predicate>cno-cnoId</predicate>
            <type>1</type>
            <pathGlue>/univ2/course/cno</pathGlue>
      </mapping>
      <mapping>
            <predicate>cno-title</predicate>
            <type>0</type>
            <pathGlue>/univ2/course</pathGlue>
            <firstArg>cno</firstArg>
            <secondArg>title</secondArg>
      </mapping>
      <mapping>
            <predicate>cno-credit</predicate>
            <type>0</type>
            <pathGlue>/univ2/course</pathGlue>
            <firstArg>cno</firstArg>
            <secondArg>credit</secondArg>
      </mapping>
<root>
```

F.3 University 3

```
<root>
      <mapping>
            <predicate>id-idId</predicate>
            <type>1</type>
            <pathGlue>/univ3/student/id</pathGlue>
      </mapping>
      <mapping>
            <predicate>id-name</predicate>
            <type>0</type>
            <pathGlue>/univ3/student</pathGlue>
            <firstArg>id</firstArg>
            <secondArg>name</secondArg>
      </mapping>
      <mapping>
```

```
            <predicate>id-major</predicate>
            <type>0</type>
            <pathGlue>/univ3/student</pathGlue>
            <firstArg>id</firstArg>
            <secondArg>major</secondArg>
      </mapping>
      <mapping>
            <predicate>id-cno</predicate>
            <type>0</type>
            <pathGlue>/univ3/student</pathGlue>
            <firstArg>id</firstArg>
            <secondArg>classes/cno</secondArg>
      </mapping>
      <mapping>
            <predicate>cno-cnoId</predicate>
            <type>1</type>
            <pathGlue>/univ3/course/cno</pathGlue>
      </mapping>
      <mapping>
            <predicate>cno-title</predicate>
            <type>0</type>
            <pathGlue>/univ3/course</pathGlue>
            <firstArg>cno</firstArg>
            <secondArg>title</secondArg>
      </mapping>
      <mapping>
            <predicate>cno-credit</predicate>
            <type>0</type>
            <pathGlue>/univ3/course</pathGlue>
            <firstArg>cno</firstArg>
            <secondArg>credit</secondArg>
      </mapping>
<root>
```