LEONE, JASON STEVEN, M.S. A Distributed Approach to XML Interoperability. (2007)
Directed by Dr. Fereidoon Sadri. 47pp.

We expand upon previous research performed on creating a lightweight infrastructure for the purpose of achieving interoperability among XML data sources. The approach is based on enriching local sources with semantic declarations so as to enable interoperability. These declarations capture the information content and concepts of the sources through mappings to a common application specific vocabulary. We design and implement a peer-to-peer network architecture through which global queries are initiated and responded to. We further examine and implement a methodology for merging the XML results of global queries into a single XML formatted response.

# A DISTRIBUTED APPROACH TO XML INTEROPERABILITY

By

Jason Steven Leone

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

## TABLE OF CONTENTS

## LIST OF FIGURES

# CHAPTER I

# INTRODUCTION

Although Extensible Markup Language's (XML) roots lie in document management, its simplicity and widespread use have led to its adoption as one of the most popular contemporary data exchange technologies. XML provides certain advantages over traditional relational database systems in this regard. Its self-documenting, human readable data format makes it easy to use. Its open, plain-text format makes it application neutral in a world largely driven by expensively licensed proprietary data systems. Having a flexible structure, XML can be employed to solve a wide variety of demands for data storage or configuration.

With such rising use of XML, particularly on the Internet, it is natural that researchers would seek to leverage the technology into areas of interoperability and data integration. The idea of combining similar data resources residing across different locations to derive an expansive body of information is not new and has indeed been researched extensively. Alternatively, having the ability to query across such a distributed network of data sources to obtain composite results is an area of database research that has also been explored considerably.

Attempting to apply this research onto the relatively new venue of XML presents some unique challenges due to the technology's non-relational nature. The flexibility of XML's syntactic model presents two main problems: (i) local data sources can model globally similar data in entirely differing ways, and (ii) local data sources may even refer to globally similar data using vastly differing terminology [3].

Various approaches to interoperability of XML data have been considered which attempt to overcome these problems through diverse means and with differing degrees of success. One straightforward approach is to force all local data sources to adhere to the same model. Though this approach would be able to draw on a very large body of traditional relational database theory and research, it would require one universal authority with control over all of the remote data sources. This is entirely impractical when the international and even inter-institutional nature of today's global networks is considered.

Another approach, defined by Halevy et al [1], is for a network of participating data sources to maintain source-to-source mappings which would allow data or queries from one source to be translated to the model of any other source. More recent work on this concept has evolved it into a peer data management system where local sources maintain semantic mappings to similarly modeled neighbors, instead of mappings for all peers [5]. Queries issued at one source can traverse semantic paths of mappings to retrieve data from any peer that is reachable across the network. This approach is further discussed in the related work section of this paper.

The approach taken by Lakshmanan and Sadri [3] is to use a lightweight infrastructure, inspired by the semantic web initiative, to enrich local data sources with semantic declarations. These declarations map local data models to a common, application specific vocabulary, which can then be used by the participant sources to issue global queries.

This thesis expands upon the existing body of work done under this approach. Firstly, methodologies for merging the results of global queries into a single XML formatted response are examined, and one such methodology is implemented. Secondly, peer-to-peer network clients are designed and implemented, through which global queries are initiated and responded to. These components are then integrated with other existing portions of the overall system already designed by previous researchers.

# CHAPTER II

# PREVIOUS WORK

The previous work that this thesis relies upon can be divided into two categories: XML technologies and XML interoperability research. The XML technologies utilized in this thesis include DTD, XPath, and XQuery. The XML interoperability research this thesis builds upon is that originally conducted by Lakshmanan and Sadri, as presented in their paper, “Interoperability on XML Data”, and further extended by the research efforts of their students.

## XML Technologies

XML data itself is not required to adhere to any sort of format or schema unless it explicitly states that it will within its containing document. The oldest schema format, Document Type Definition (DTD), is included in the XML 1.0 W3C Recommendation [10]. It is derived substantially from the Standard Generalized Markup Language (SGML). A DTD is a group of markup declarations which together define a grammar for a class or set of documents by means of constraining the structure of those documents. Two common markup declarations are element and attribute-list declarations. Element declarations specify the allowed elements within a document, and the orientation of those elements relative to each other (ordering, containment, etc). Attribute-lists specify the allowable set of attributes for each of the specified elements. An XML document is said to be valid if it complies with the constraints specified in its associated DTD.

XML Path Language (XPath), defined in the XPath 1.0 W3C Recommendation, is a language for addressing parts of an XML document [12]. This addressing is accomplished by means of a path expression, a sequence of location steps each separated by a ‘/’ character. These path expressions are used for matching path patterns, where an XML document is considered to be modeled as a tree of nodes. XPath defines seven such types of nodes, including element, attribute, and text nodes. A path expression can be evaluated to yield an object of type node-set, Boolean, number, or string. Path expressions also provide some functionality to manipulate strings, numbers, and Booleans, primarily through the inclusion of predicates within the expression.

XQuery, defined in the XQuery 1.0 W3C Recommendation, is a query language designed to query XML data; it provides a way to extract and manipulate data from XML (compatible) data sources [13]. XQuery is built upon XPath. Hence, it also considers XML data to be modeled as trees of nodes, and XPath expression syntax can be used for addressing parts of the XML data. XQuery queries are said to be similar to SQL queries in appearance, and are organized into “FLWOR” expressions. “FLWOR” expressions are composed of five sections: for, let, where, order by, and return. The output of an XQuery can be XML; this powerful functionality allows new XML (documents) to be formed as a result of an XQuery that runs across potentially several XML data sources.

## XML Interoperability Research

Lakshmanan and Sadri, in their paper “Interoperability on XML Data” [3] examine how XML can be utilized in achieving goals of interoperability and data integration. They suggest (inspired by the Semantic Web Initiative) that, although XML by itself cannot accomplish interoperability, by enriching the XML data with semantic declarations, the results would be interoperable within the target application area. This overcomes the two key problems of XML data integration that they identify: heterogeneous data modeling and heterogeneous terminology.

To begin with, the authors describe a lightweight infrastructure for enabling interoperability across multiple data sources. Interoperability is to be achieved via common ontologies. Each local source would retain its data in whatever native model it chooses. If the local administrator desires to participate in data sharing under the system, he would select a set of properties (which seem to most closely fit his local data) from one of the ontologies and then provide mappings of his local data onto those properties. These mappings have a structure similar to the binary relation format specified by the RDF, or Resource Definition Framework. In RDF, two resources or objects are connected together in a binary relation based upon a property which they have in common. A collection of these relations can be seen to serve as a traditional binary table structured view over a tree-like structured XML data set.

Once these mappings are in place, user queries can be posed using the common terminology of the ontology properties. A “coordinator” would be responsible for query expansion, preparation, optimization, and transformation, as well as coordinating sub-query execution at the different sources and combining the results of the sub-queries. This accounts for cases where the global query is broken down into local sub-queries and/or inter-source queries. In a local sub-query, all of the data necessary to answer the sub-query resides at the same source. In an inter-source query, data from different sources must be joined together to answer the query. Figure 2.1 illustrates the conceptual architecture of this system.

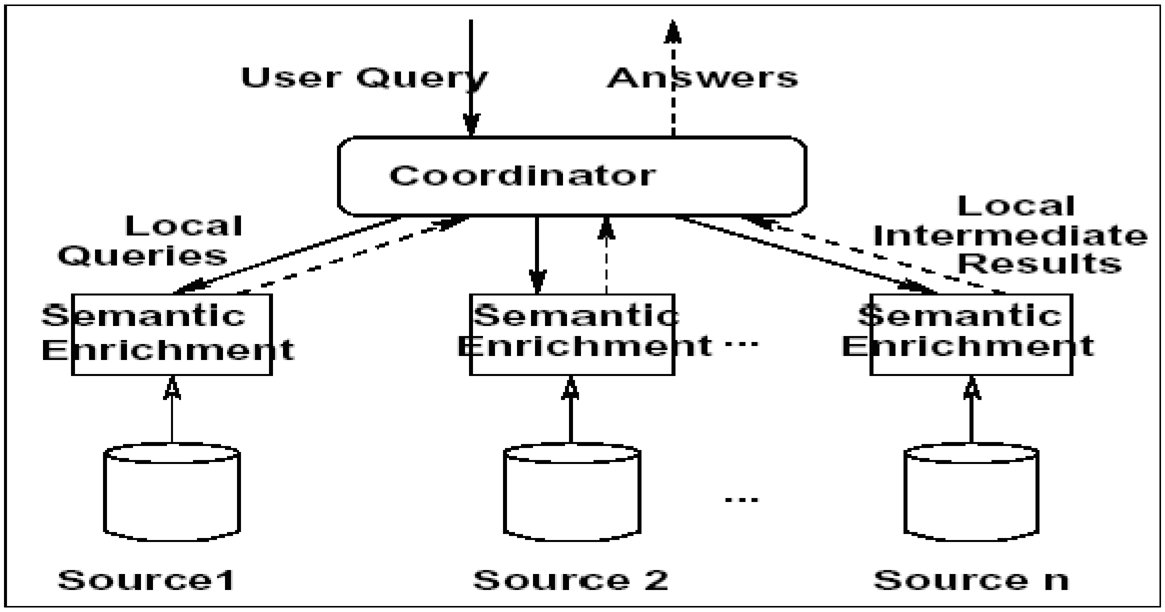


*Figure 2.1: Conceptual System Architecture*

The authors suggest that the mappings can be constructed with the help of tools, or by hand. Figure 2.2 shows the format of a mapping for a binary property p. In the case of XML data, it is helpful to visualize a tree representation of the data in order to understand how the mappings might be accomplished. In the figure, there are three paths specified. The first path is for the “glue” variable, represented by $G. This path is a common ancestor node, usually the least common ancestor, for the other two nodes being bound in the relation. The second path, represented by $X, and the third path, represented by $Y, are the two XML element nodes (or attributes) tied to the property. Consistent with the RDF specification [9], f($X) is a URI-generating function, “a one-to-one function that takes an identifier of an object and generates a unique URI for that object” [3]. Similarly but less restrictive, g($Y) can be either a URI-generating function or $Y itself.

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

*Figure 2.2: Binary Property Mapping Format*

A global query initiated over the participant data sources, i.e. those data sources which have specified the relevant mappings, will be translated using the mappings to queries appropriate for each of the local sources. The translation algorithm (noted as “Algorithm 1”) as specified by Lakshmanan and Sadri is as follows [3]:

**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.
(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.

Again, the authors envision a coordinator that would dispatch queries to the local sources, receive partial results back from the local sources, and ultimately combine the results to achieve a meaningful answer to the query. Some efficiency can be gained by having the local sources execute their queries in parallel.

# CHAPTER III

# CONTRIBUTIONS

The goal of this thesis is to build upon the approach taken in [3] and expand upon the related body of work accomplished by previous researchers. We modify the conceptual system architecture (see figure 2.1) to design and implement the system as a peer-to-peer network; instead of queries and results passing through a single coordinator, in our network, any peer can pose queries and receive the results of those queries. Queries are posed in the form of XQuery, and results are received in the form of XML data. In order to process the results of queries, XML merging techniques are explored, and one such algorithm is implemented.

## XML Merging

When a peer in the distributed network issues a query, it expects to receive responses to its query from the other peers over a random amount of time. As it receives these responses, it needs to combine them together, along with its own local query results, to obtain a meaningful answer to the global query. This thesis assumes the responses from the network peers will all be formatted as XML data, valid against some predetermined global “answer” DTD (see the Future Work section for more on this notion). The peer that issued the query must then be able to merge together the XML data (results) it receives from the other peers. To this end we have explored various approaches and have developed an algorithm for merging XML.

## Schema-less Merging

Our first approach to merging was to attempt to merge the XML data together without any guidance from a schema or DTD. We call this approach Schema-less Merging. The technique is to consider the XML data as a tree-like data structure, and begin by attempting to merge the roots of two XML documents (trees), without regard for the children of the roots. This involves confirming equality between the root nodes' element types, text values (if present), and attribute lists. Attributes that are present in both roots must have equal values; attributes present in one root but not the other can simply be added to the merged root node. If the two roots can be merged without regard for their children, then an attempt can be made to merge their children together. This amounts to taking the children of one root one at a time and attempting to merge them with any of the children of the other root. This is done with a recursive application of the algorithm. If the currently considered child is successfully merged with one of the second root's children, then the next of the first root's children can be considered. If the child is not successfully merged with any of the second root's children, it is simply added as a child to the merged root.

Although this merging technique works well in some limited cases, for the most part it provides meaningless results. The problem is that the algorithm never fails to merge two XML documents, because when it cannot merge particular nodes, it simply ends up adding them individually to the resultant tree. The merged result has no sorts of constraints or consistent nature, even when merging multiple pairs of documents which all adhere to the same schema. For example, merging XML data which contains the first and last names of people as children elements of a "person" node may in some cases result in "person" nodes which have multiple first names and single last names, or single first names and multiple last names, or even multiple first and last names. The recursive concept seemed promising however, and when combined with some techniques to consider a schema in the merging process, it led to our next approach which proved to be more successful.

## DTD Driven Merging

Our next approach to merging is based on the concept of using a Document Type Definition to guide the process of merging two trees of XML data. We again attempt to merge the nodes of the tree in a recursive manner, but this time we use the element content models defined within the DTD as a reference for where and how element nodes may appear within the resultant merged tree. This provides the consistency and constraints that were lacking from the Schema-less Merging approach.

DTD allows for some considerably complex possibilities in defining an XML schema. In order to make the merging process more manageable, we assume a simplified, yet still valid, DTD model. Specifically, the DTD model used in the following merging approach is constrained as follows:

- A DTD can only have <!ELEMENT> and <!ATTLIST> markup declarations.
- An element’s content model must be declared to contain other elements or be of PCDATA or CDATA types only.

- An element’s content model may not contain parenthesized subsections.
- An element’s content model may not contain choice lists (use of “|”).
- An element’s content model may contain the optional character operators following element names (use of “+”, “*”, “?”).
- Attributes must be of type CDATA.
- An attribute’s default declaration must be #IMPLIED.

Under the assumption of this simplified DTD model, the MergeTrees algorithm is used to merge two XML documents. Here, the term ‘XML document’ is used to refer to any well-formed XML data that conforms to a specified DTD, not necessarily a file system document. The MergeTrees algorithm considers XML data as a tree of nodes, where each node corresponds to a single element defined within the related DTD. The main idea of the algorithm is to merge the two XML documents while referring to the associated DTD as a guide to how the documents can be appropriately combined. The algorithm is specified below:

**Algorithm MergeTrees**
Input: DTD document D; nodes T1 and T2 from XML documents 1 and 2 respectively. Documents 1 and 2 are assumed to be valid against D. To fully merge the two documents, T1 and T2 would be the roots of the two documents.
Output: The new node Tm that is the merge of T1 and T2, or an indication (null) that the two nodes could not be merged. Tm is valid against D.
Method:
1. If T1 and T2 are of different element types or have different text values then return null. Otherwise, set Tm’s type and value to T1’s type and value.
2. Attempt to MergeAttributes with inputs Tm, T1, and T2. If MergeAttributes returns false then return null.
3. Attempt to MergeChildren with inputs D, Tm’s element type, T1’s list of children, T2’s list of children. If MergeChildren returns null then return null. Otherwise, set Tm’s list of children to the result of MergeChildren and return Tm.

Algorithm MergeTrees makes calls to MergeAttributes and MergeChildren. MergeAttributes will merge the attribute lists of two nodes whose type is assumed to be the same. MergeChildren will merge the ordered list of children of two nodes whose type is again assumed to be the same. Both algorithms are listed below:

**Algorithm MergeAttributes**
Input: Node Tm, the node to contain the merged attributes; nodes T1 and T2, whose attributes are to be merged.
Output: True if T1's and T2's attributes were successfully merged and set into Tm; false otherwise.
Method:

1. If T1 and T2 have no attributes, then set Tm's attributes to null and return true.
2. If T1 or T2 has no attributes, then set Tm's attributes equal to those of the node which does have attributes and return true.
3. Look at each attribute in T2. If it is not an attribute of T1, set it as an attribute of Tm. If it is an attribute of T1 and has a different value than that attribute in T1, then return false.
4. Set all attributes of T1 as attributes of Tm and return true.

**Algorithm MergeChildren**
Input: DTD document D; element type E of the parent node; ordered list L1 of a nodes children; ordered list L2 of a nodes children. E is assumed to be a valid element type within D, L1 and L2 are assumed to be valid lists of child elements for a parent of type E, within D.
Output: An ordered list Lm that is the merge of L1 and L2, or an indication (null) that the two lists could not be merged. Lm is a valid list of child elements for a parent of type E, within D.
Method:

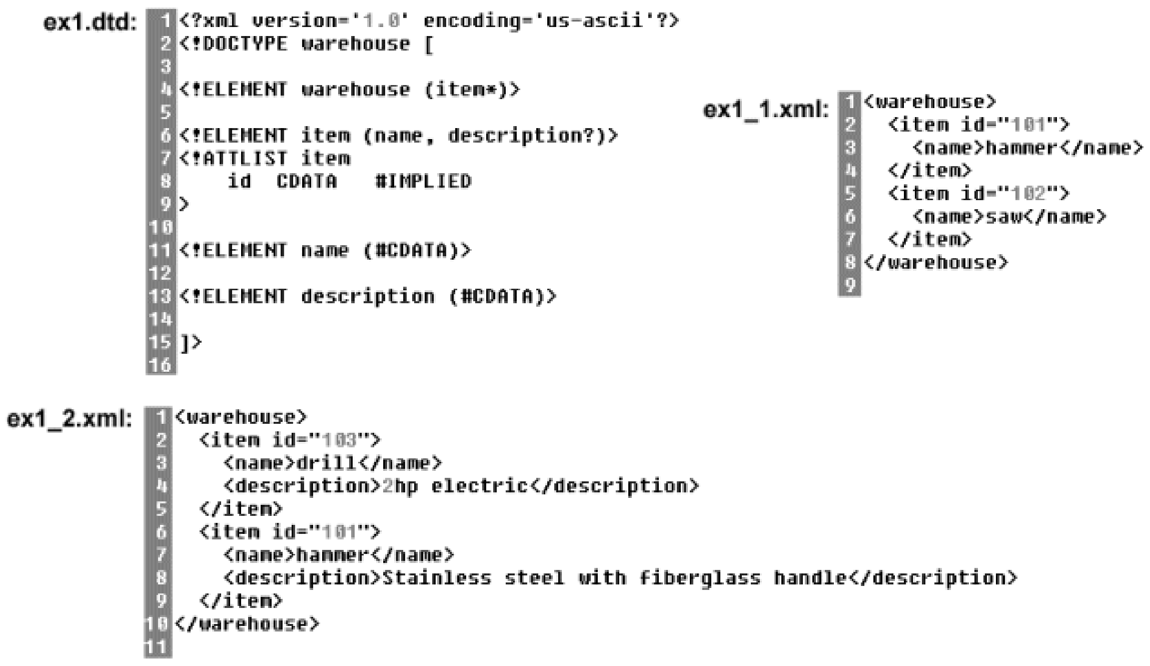
1. Create a pointer Pe = 0, to sequentially parse through (a copy of) the content model of E (found by looking up E in D) represented as a list Le.
2. Create pointers P1 = 0 and P2 = 0, to sequentially parse through L1 and L2.
3. Create an empty ordered list Lm.
4. If Pe is at the end of Le, but P1 or P2 is not at the end of its respective list, then return null.
5. If Pe, P1, and P2 are all at the end of their respective lists, then return Lm.

6. If P1 and P2 are at the end of their respective lists, then parse the remainder of Le to determine if there are any required child elements remaining. If so, return null. Otherwise, return Lm.
7. If P1 is at the end of L1, but P2 is not at the end of L2 then compare Le[Pe] to L2[P2]:
    1. If Le[Pe] is a required element (i.e. has no optional character operator), and L2[P2] is not of the correct element type, then return null. Otherwise, add L2[P2] to Lm, increment Pe and P2, and return to step 4.
    2. If Le[Pe] is marked with the character operator '+', modify Le to replace the 'element+' entry with the two entries 'element, element*'. Return to step 4.
    3. If Le[Pe] is marked with the character operator '*', then if L2[P2] is of the correct element type, add L2[P2] to Lm and increment P2. Otherwise, if L2[P2] is not of the correct element type, increment Pe. Return to step 4.
    4. If Le[Pe] is marked with the character operator '?', then if L2[P2] is of the correct element type, add L2[P2] to Lm and increment P2 and Pe. Otherwise, if L2[P2] is not of the correct element type, increment Pe. Return to step 4.
8. If P2 is at the end of L2, but P1 is not at the end of L1, then handle similar to step 7.
9. None of Pe, P1, or P2 is at the end of its respective list. Look at Le[Pe].
10. If Le[Pe] is a required element, then look at L1[P1] and L2[P2]:
    1. If both are of the correct element type, then attempt to MergeTrees with inputs D, L1[P1], L2[P2]. If the merge is successful, add the result to Lm and increment Pe, P1, P2. Otherwise add L1[P1] to Lm and increment Pe, P1. Return to step 4.
    2. If L1[P1] is of the correct element type but L2[P2] is not, then add L1[P1] to Lm and increment Pe and P1. Return to step 4.
    3. If L2[P2] is of the correct element type but L1[P1] is not, then add L2[P2] to Lm and increment Pe and P2. Return to step 4.
    4. If neither are of the correct element type, then return null.
11. If Le[Pe] is marked with the character operator '+', then do the following:
    1. Make a list S1 of the next consecutive elements in L1 that are of the correct element type (Le[Pe]). Advance P1 past this consecutive group.
    2. For each of the next consecutive elements in L2 that are of the correct element type, attempt to merge them one by one into S1 using MergeTrees. Any that are unsuccessfully merged into S1 are appended to the end of S1. Advance P2 past this consecutive group.
    3. If S1 is of size 0, return null.
    4. Add the elements of S1 to Lm, increment Pe, and return to step 4.
12. If Le[Pe] is marked with the character operator '*', then handle similarly to step 11, omitting the check of step 11.3.

13. If Le[Pe] is marked with the character operator ‘?’, then look at L1[P1] and L2[P2]:
    1. If both are of the correct element type, then attempt to MergeTrees with inputs D, L1[P1], L2[P2]. If the merge is successful, add the result to Lm and increment Pe, P1, P2. Otherwise add L1[P1] to Lm and increment Pe, P1. Return to step 4.
    2. If L1[P1] is of the correct element type but L2[P2] is not, then add L1[P1] to Lm and increment Pe and P1. Return to step 4.
    3. If L2[P2] is of the correct element type but L1[P1] is not, then add L2[P2] to Lm and increment Pe and P2. Return to step 4.
    4. If neither are of the correct element type, then increment Pe. Return to step 4.

## DTD Driven Merging Example

We will walk through an example of the merge of two XML documents to demonstrate the algorithms defined for the DTD Driven Merging approach. Consider the two XML documents and their corresponding DTD presented in figure 3.1. As a shorthand notation for our example, we will refer to the MergeTrees algorithm as MT, the MergeAttributes algorithm as MA, and the MergeChildren algorithm as MC.


ex1.dtd:

```
<?xml version='1.0' encoding='us-ascii'?>
<!DOCTYPE warehouse [

<!ELEMENT warehouse (item*)>

<!ELEMENT item (name, description?)>
<!ATTLIST item
    id  CDATA   #IMPLIED
>

<!ELEMENT name (#CDATA)>

<!ELEMENT description (#CDATA)>

]>
```

ex1_1.xml:

```
<warehouse>
  <item id="101">
    <name>hammer</name>
  </item>
  <item id="102">
    <name>saw</name>
  </item>
</warehouse>
```

ex1_2.xml:

```
<warehouse>
  <item id="103">
    <name>drill</name>
    <description>2hp electric</description>
  </item>
  <item id="101">
    <name>hammer</name>
    <description>Stainless steel with fiberglass handle</description>
  </item>
</warehouse>
```

*Figure 3.1: Example XML Documents and Corresponding DTD*

We start by invoking MergeTrees with the inputs of ex1.dtd as $D_1$, the <warehouse> node of ex1_1.xml as $T1_1$, and the <warehouse> node of ex1_2.xml as $T2_1$. In step MT.1, $T1_1$ and $T2_1$ are of the same element type and have no text values, so $Tm_1$ is set to be an element of type "warehouse". In step MT.2, a call is made to MergeAttributes with inputs $Tm_1$, $T1_1$, and $T2_1$. This leads to step MA.1, where $T1_1$ and $T2_1$ have no attributes so $Tm_1$'s attributes are set to null (none) and MergeAttributes exits returning a value of true. Figure 3.2 shows an abstract visualization of the merge so far.

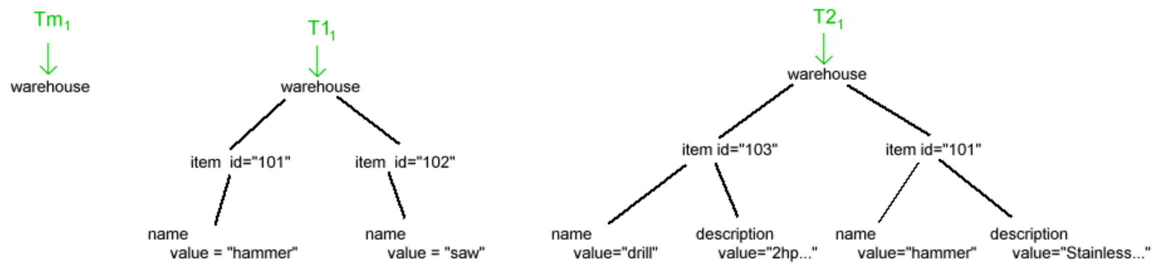


*Figure 3.2: Visualization of Merging Steps, Part 1*

Next is step MT.3, where MergeChildren is called with the inputs of $D_1$ as $D_2$, "warehouse" as $E_1$, $T1_1$'s list of children as $L1_1$, and $T2_1$'s list of children as $L2_1$. Steps MC.1 and MC.2 create the pointers $Pe_1$, $P1_1$, and $P2_1$ which all point to the starts of their respective lists. Figure 3.3 shows an abstract visualization of the merge at this point.

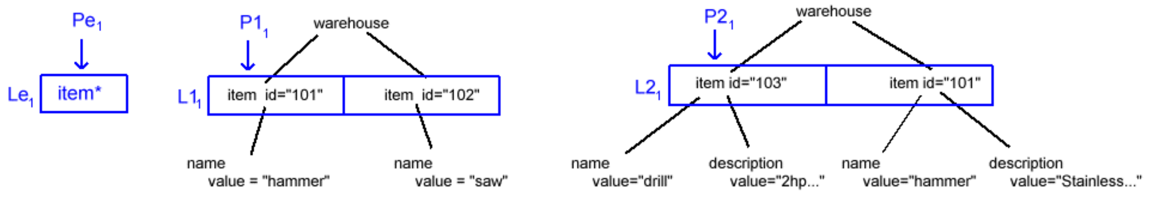


*Figure 3.3: Visualization of Merging Steps, Part 2*

Step MC.3 creates the empty list $Lm_1$. Steps MC.4 through MC.8 do not apply, so in step MC.9 $Le_1[Pe_1]$ is examined, and we see that it indicates "item" elements qualified with the '*' character operator. This means that 0 or more <item> nodes are expected to appear next. Step MC.12 matches this condition, so step MC.12.1 is executed, creating the list $S1_1$ and advancing the pointer $P1_1$. Figure 3.4 visualizes the merge state so far.

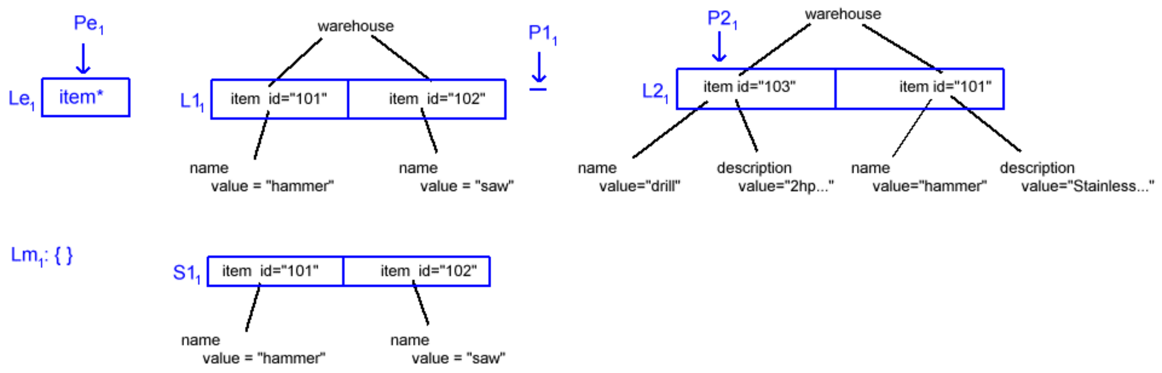


*Figure 3.4: Visualization of Merging Steps, Part 3*

Step MC.12.2 begins by attempting to merge $S1_1[0]$ with $L2_1[P2_1]$. The subsequent call to MergeTrees (inputs: $D_2$, $S1_1[0]$, $L2_1[P2_1]$) passes through step MT.1 ok, but fails to merge the two nodes' attributes in step MT.2 since they have differing "id" values. Step MC.12.2 continues by attempting to merge $S1_1[1]$ with $L2_1[P2_1]$. The subsequent call to MergeTrees (inputs: $D_2$, $S1_1[1]$, $L2_1[P2_1]$) also fails in step MT.2 since the "id's" again differ. Since $L2_1[P2_1]$ cannot be merged into $S1_1$, it is marked to be added onto the end of $S1_1$, and $P2_1$ is advanced. Figure 3.5 shows an abstract visualization of the merge at this point.

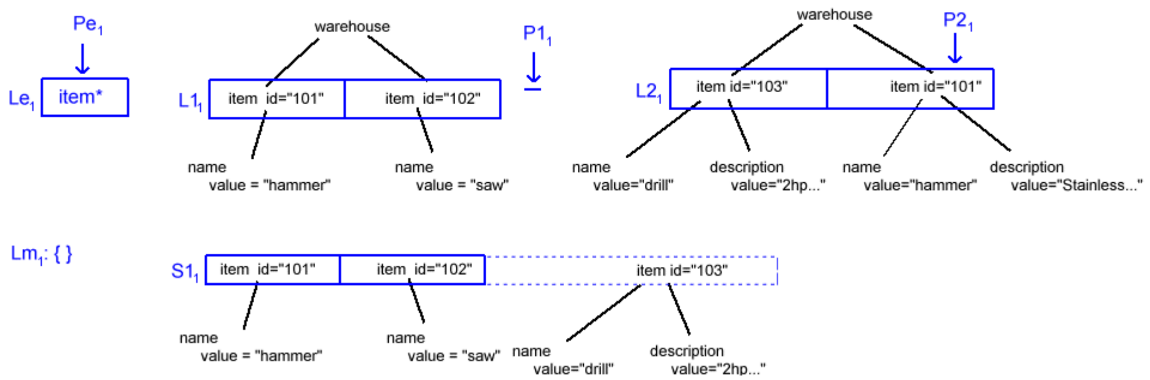


*Figure 3.5: Visualization of Merging Steps, Part 4*

Step MC.12.2 continues by attempting to merge $S1_1[0]$ with $L2_1[P2_1]$ which results in a call to MergeTrees with inputs $D_2$ as $D_3$, $S1_1[0]$ as $T1_2$, and $L2_1[P2_1]$ as $T2_2$. In step MT.1, $T1_2$ and $T2_2$ are of the same element type and have no text values, so $Tm_2$ is set to be an element of type "item". In step MT.2, a call is made to MergeAttributes with the inputs $Tm_2$, $T1_2$, and $T2_2$. Steps MA.1 and MA.2 do not apply. In step MA.3, the "id" attribute of $T2_2$ is compared to the "id" attribute of $T1_2$ and seen to have the same value, so it is ignored. In step MA.4, the "id" attribute of $T1_2$ is set to be an attribute of $Tm_2$, and MergeAttributes exits returning a value of true. Figure 3.6 show an abstract visualization of the merge so far within the current call to MergeTrees.

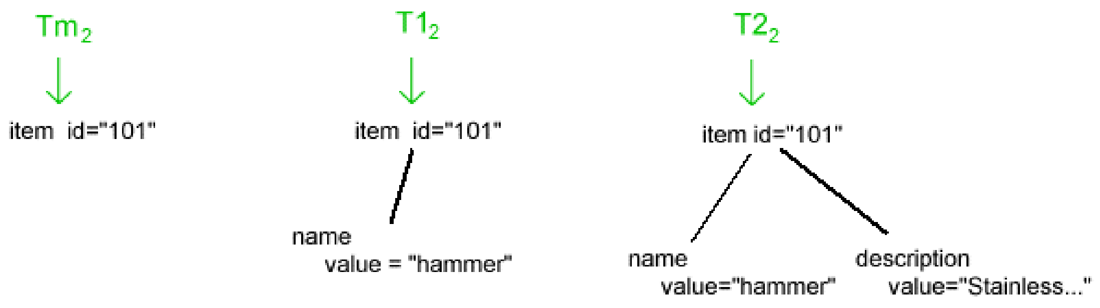


*Figure 3.6: Visualization of Merging Steps, Part 5*

Step MT.3 is next, in which a call to MergeChildren is made with $D_3$ as $D_4$, "item" as $E_2$, $T1_2$'s list of children as $L1_2$, and $T2_2$'s list of children as $L2_2$. Steps MC.1 and MC.2 create the pointers $Pe_2$, $P1_2$, and $P2_2$ which all point to the starts of their respective lists. Figure 3.7 visualizes this merge state.

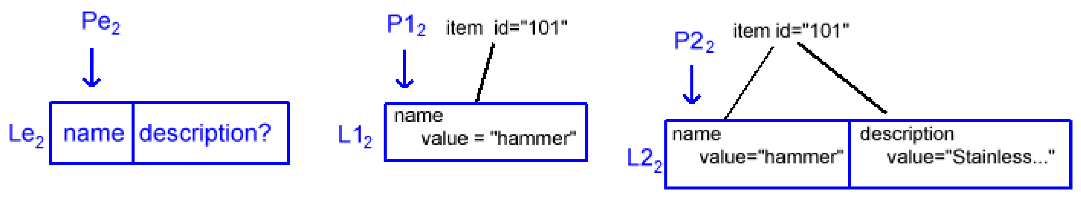


*Figure 3.7: Visualization of Merging Steps, Part 6*

Step MC.3 creates the empty list $Lm_2$. Steps MC.4 through MC.8 do not apply, so in step MC.9 $Le_2[Pe_2]$ is examined, and we see that it indicates an unqualified "name" element. This means that exactly one <name> node is expected to appear next. Step MC.10 matches this condition, so $L1_2[P1_2]$ and $L2_2[P2_2]$ are examined. In step MC.10.1, both nodes are of the correct element type, so MergeTrees is invoked with inputs $D_4$ as

$D_5$, $L1_2[P1_2]$ as $T1_3$ and $L2_2[P2_2]$ as $T2_3$. Steps MT.1 and MT.2 result in $Tm_3$ being set to an element of type "name" with a value of "hammer". Step MT.3 invokes MergeChildren with inputs of $D_5$, "name", $T1_3$'s (empty) list of children, and $T2_3$'s (empty) list of children. This run of MergeChildren exits at step MC.5 returning an empty list which is set to be $Tm_3$'s list of children. MergeTrees then returns $Tm_3$, which causes the calling step MC.10.1 to resume (of the call MergeChildren( $D_3$, "item", {$T1_2$->name}, {$T2_2$->name, $T2_2$->description} )). $Tm_3$ is added to $Lm_2$, and $Pe_2$, $P1_2$, $P2_2$ are all incremented, and the algorithm goes to step MC.4. Figure 3.8 visualizes the merge state at this point.

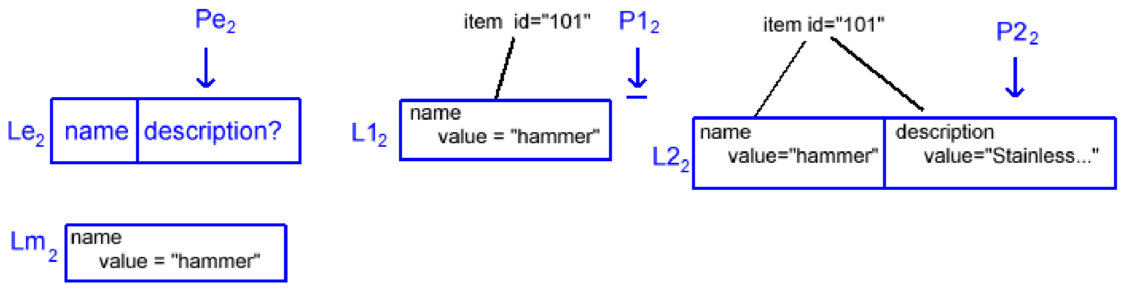


*Figure 3.8: Visualization of Merging Steps, Part 7*

Steps MC.4, MC.5, and MC.6 do not apply. In step MC.7, $P1_2$ is at the end of $L1_2$, so $Le_2[Pe_2]$ and $L2_2[P2_2]$ are examined. Step MC.7.4 applies, and $L2_2[P2_2]$ is of the correct (expected) element type, so $L2_2[P2_2]$ is added to $Lm_2$, and $Pe_2$ and $P2_2$ are both incremented. The algorithm moves to step MC.4. Figure 3.9 shows a visualization of the merge stat at this point.

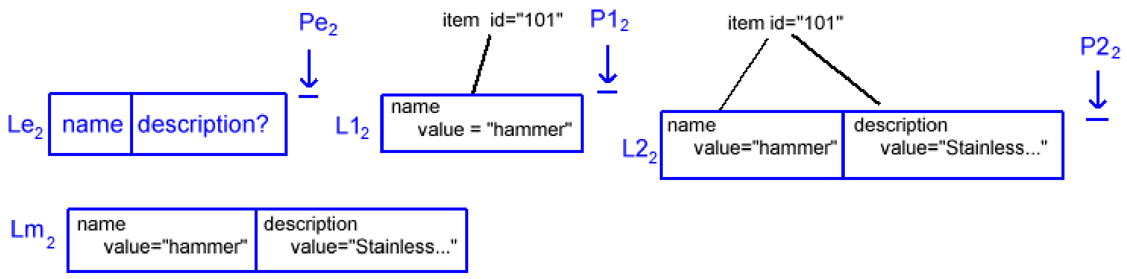


*Figure 3.9: Visualization of Merging Steps, Part 8*

Step MC.4 does not apply, but step MC.5 does, so $Lm_2$ is returned by MergeChildren. The calling step MT.3 resumes (of the call MergeTrees( $D_2$, $S1_1$[0], $L2_1$[$P2_1$] )), where $Tm_2$'s list of children is set to $Lm_2$. Figure 3.10 shows a visualization of this result.

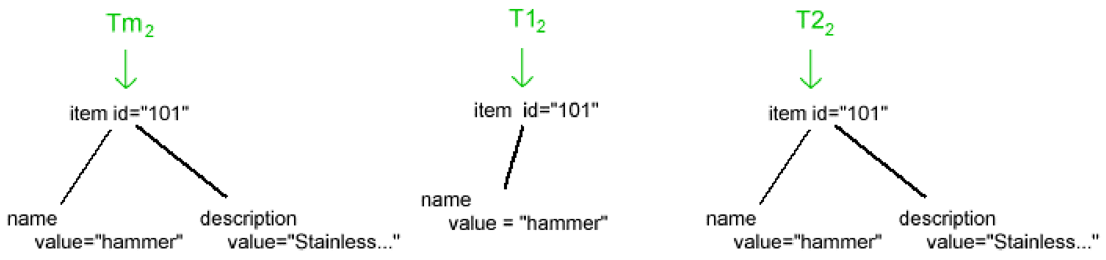


*Figure 3.10: Visualization of Merging Steps, Part 9*

$Tm_2$ is returned, and Step MC.12.2 resumes (of the call MergeChildren( $D_1$, "warehouse", {$T1_1$->item(id="101"), $T1_1$->item(id="102")}, {$T2_1$->item(id="103"), $T2_1$->item(id="101")} )) with $S1_1$[0] being set to $Tm_2$ and $P2_1$ being incremented. Figure

3.11 shows an abstract visualization of the merge so far within the current call to MergeChildren.

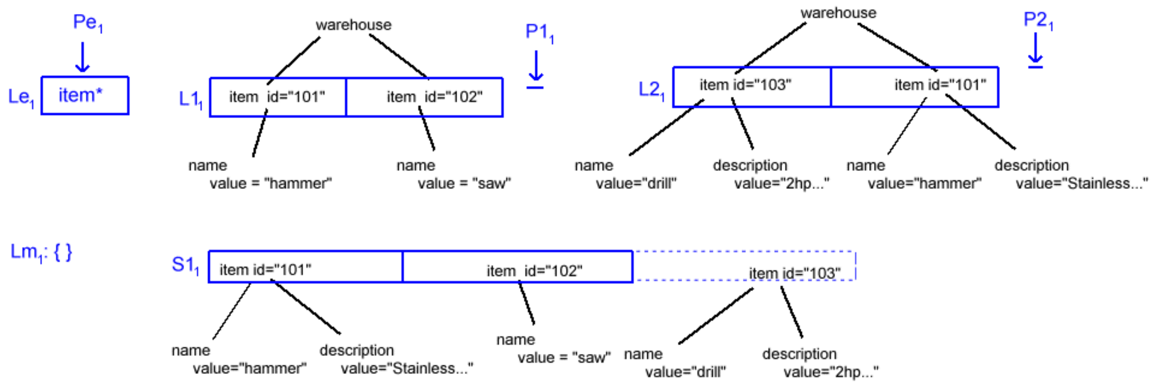


*Figure 3.11: Visualization of Merging Steps, Part 10*

Step MC.12.3 is omitted. Step MC.12.4 adds the elements of $S1_1$ to $Lm_1$ and increments $Pe_1$ before moving to step MC.4. Figure 3.12 shows a visualization of the merge at this point.

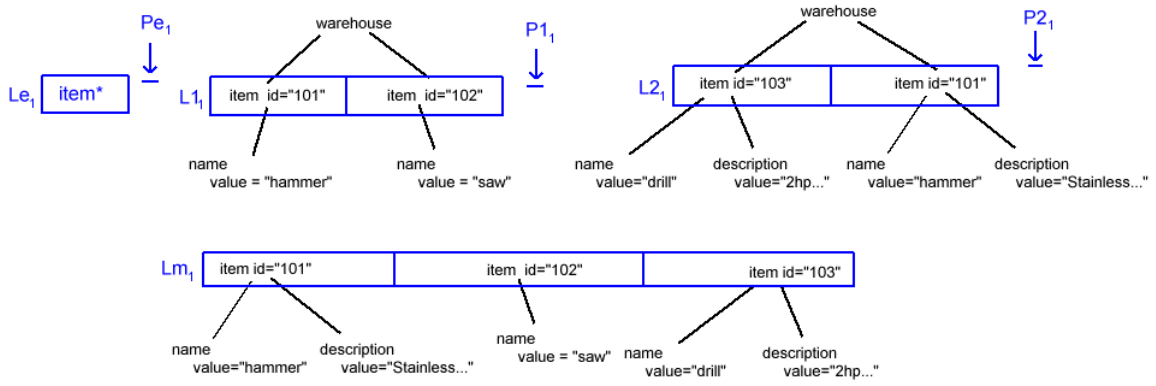
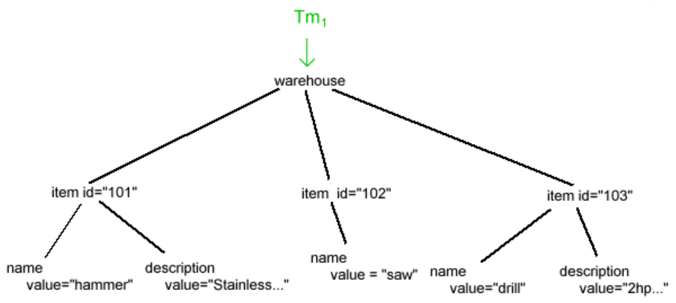
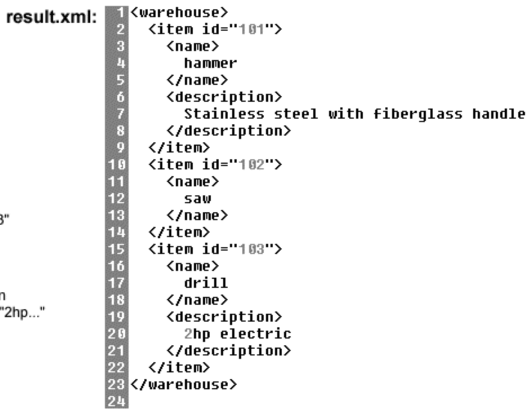


*Figure 3.12: Visualization of Merging Steps, Part 11*

Step MC.4 does not apply, but step MC.5 does, so Lm$_1$ is returned by MergeChildren. The calling step MT.3 resumes (of the call MergeTrees( ex1.dtd, ex1_1.xml:<warehouse>, ex1_2.xml:<warehouse> )), where Tm$_1$'s list of children is set to Lm$_1$. Tm$_1$ is returned, and the merge is complete, as illustrated in figure 3.13.




result.xml:

```
<warehouse>
  <item id="101">
    <name>
      hammer
    </name>
    <description>
      Stainless steel with fiberglass handle
    </description>
  </item>
  <item id="102">
    <name>
      saw
    </name>
  </item>
  <item id="103">
    <name>
      drill
    </name>
    <description>
      2hp electric
    </description>
  </item>
</warehouse>
```

*Figure 3.13: Merge Result*

## Distributed Network Design and Overview

Prior to our efforts, a network implementation of the conceptual system architecture (see Figure 2.1) had not been done. In considering an implementation of the system, we had one primary architectural change we wanted to introduce: the removal of the single system coordinator. The coordinator represented a single point where user queries would be posed, and the answers to those queries would be given to the user. Charged with the responsibilities of query expansion, preparation, optimization, transformation, execution coordination, and result aggregation, the coordinator was effectively a centralized authority residing over the system. This brought to the system all of the traditional problems of centralized management, including administration and configuration. The modified system architecture is purely peer-to-peer. Queries can be initiated from any client; that client is responsible for result aggregation as it receives results from other clients. Query translation is expected to occur at each client since it has complete knowledge of its own data modeling and semantic mappings onto the global ontology. Issues of optimization are not currently addressed by our system, but it is envisioned that a coordination/optimization component would be added to each client in the future (see the Future Work section).

One of our key goals in designing the network was to produce a system that will be used by other researchers for future development, testing, and other continuing work under the Lakshmanan and Sadri approach to interoperability on XML data. To support this goal we emphasized extendibility and flexibility in the design, where possible.

As the system is expected to reside on the Internet (or an intranet), there were no specific network hardware design or even application-hardware interaction issues to consider; use of a sufficiently advanced programming language (such as Java or C++) allows the developer to rely on the language's API to provide functionality for establishing connections between clients on an existing (physical) network, and/or transmitting data across a network. Left for the developer is the design of the data to be transmitted and what meaning that data has, as well as the determination of when communications are to take place. In our design, all communications between clients are encapsulated within an application layer defined object, which we have called "Packet". Hence the data transmitted across the network is representative of these Packet objects. The available logical commands which clients can issue are all defined within the Packet object; the clients themselves are designed to conditionally react to the type of command issued. This provides the system with the flexibility to handle many commands and provides an organizational structure to ease the task of extending the body of commands used.

In order to avoid complex issues of network traffic management, we adopt a peer network model in which each network client knows of the existence of all other clients on the network and is capable of contacting them directly; this contrasts with other peer network models in which network communications are relayed along paths across multiple peers in order to reach all clients in the network (see the Related Work section). Figure 3.14 illustrates our peer network model.

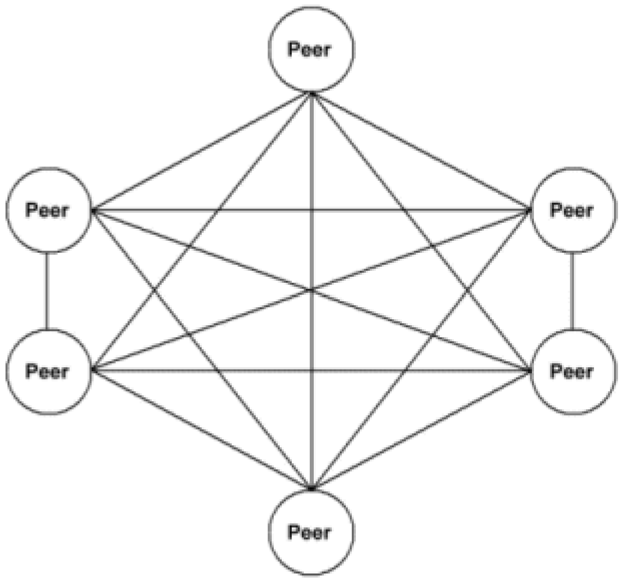


*Figure 3.14: Peer Network Model*

Communications between peers is carried out over TCP/IP connections (provided through functionality available in the programming language's API). Persistent connections are not used; when a peer needs to contact another peer, it refers to its table of known hosts to find the IP address and port number of the target peer, then connects to that peer long enough to send the desired data, closing its connection afterwards. Certainly the use of TCP/IP connections adds some overhead to the system, but the protocol's reliability and error control mechanisms guarantee a high probability that the peer's transmission will correctly reach its target. Clients can join the system by connecting to another client already on the system. The receiving client notifies all other clients of the new client, as well as informing the new client about the existing members of the network.

Each client listens on a configurable port for incoming communications. Since there is no synchronization of communications, the clients are multi-threaded to handle multiple simultaneous connections. As an incoming connection is established, the client spawns a new thread to handle that connection.

One of the tasks that clients have to accomplish is to translate received queries and run them locally. This is modularized into a "query engine" which is solely responsible for performing these duties. The engine is designed such that any of multiple translation functions can be invoked. This is another area where ease of extendibility is evident, as adding new translation schemes is straightforward, and the mechanism for calling those schemes is already available.

Each client provides functionality to initiate a query over the system. In the process of initiating a query, the client must translate and execute the query locally in addition to communicating its request with the other clients on the network. The client must keep track of the results of the query it has initiated until all results are received from the other clients, or until the query has expired against a configurable time limit. Though the client provides this query execution functionality, the client itself is entirely decoupled from any sort of user interface. This allows the client to be created and controlled through flexible means. Our system employs the Facade design pattern whereby a controller object binds a user interface to a client such that a user may interact with the system. In our testing of the system, we also created a sort of "dumb" client wrapper which simply instantiates a client to act as a data source listening on the network and provides no user interaction mechanisms.

## Implementation and Code Organization

Our system is implemented in Java under the J2SE 1.5.0 API [7]. The code was developed under two separate projects. The first project is XMLMerge, which implements the XML merging functionality of the system, primarily as specified in the MergeTrees, MergeAttributes, and MergeChildren algorithms of the DTD Driven Merging approach. The second project is XMLT, which implements the distributed network clients and user interface to the system. XMLT uses the functionality implemented in XMLMerge to handle the merging of query results. Appendix A contains class diagrams for both projects.

XMLMerge makes use of the Jakarta-ORO Java classes available from the Apache Jakarta Project website [6] as a Java jar file. This library contains a set of text-processing Java classes that provide Perl and AWK-like regular expression functionality. It also contains various utility classes for performing substitutions, splits, and other string manipulations.

XMLT makes use of the Saxon-B 8.8 Java classes available from the Saxonica website [8] as a collection of Java jar files. These libraries contain classes that provide a complete, conformant implementation of the XQuery 1.0 and XPath 1.0 W3C Recommendations. XMLT also makes use of XMLMerge via a distribution of the project as a jar file.

## XMLMerge Overview

The XMLMerge project contains the following classes:

xmlmerge.model.Attribute
xmlmerge.model.DTDDocument
xmlmerge.model.DTDElement
xmlmerge.model.ElementChild
xmlmerge.model.Node
xmlmerge.model.Quantifier
xmlmerge.model.XMLMerger

The Attribute, DTDDocument, DTDElement, ElementChild, and Quantifier classes are used to model DTD documents and their structured content. DTDDocument is capable of reading a DTD document from the local file system or parsing a string representation of a DTD. The Node class models an XML element as a node under the conceptualization of an XML document as a tree of nodes (elements).

The XMLMerger class fully implements the MergeTrees, MergeAttributes, and MergeChildren algorithms. It is capable of reading XML documents from the local file system or parsing string representations of XML data to create trees of Nodes from the XML, to be used in the merging algorithms. It uses DTDDocument for the DTD documents needed by the algorithms.

## XMLT Overview

The XMLT project contains the following classes:

xmltdistributed.model.ClientRequestHandler
xmltdistributed.model.Packet
xmltdistributed.model.QueryEngine
xmltdistributed.model.QueryRecord
xmltdistributed.model.RecordWatchThread
xmltdistributed.model.XMLTHost
xmltdistributed.network.XMLTDistributedClient
xmltgui.XMLTGUI

xmltgui.model.XMLTController

The XMLTDistributedClient implements a multi-threaded peer network client. It opens a socket and listens on a specified port waiting for incoming connections. When it receives a connection, it spawns a new ClientRequestHandler thread to process the request. It uses the QueryRecord class to track queries it initiates and poses over the network; a running result is kept within the QueryRecord, as well as information on which clients have not yet responded. The RecordWatchThread class is a background thread of XMLTDistributedClient's, which is responsible for monitoring QueryRecords and re-querying peers who have not responded within a certain time threshold, or expiring QueryRecords if they have become too old.

All communications between clients are encapsulated within Packet objects. This takes advantage of Java's features for streaming objects over sockets. The ClientRequestHandlers process Packets they receive and, depending upon their contents, they may take several different courses of action, including executing a query and transmitting its response, updating a QueryRecord with a received response, adding a new peer to the list of known hosts, and notifying the network of a new peer who joined.

The QueryEngine class is responsible for executing XQueries at the local source. It uses the Saxon-B 8.8 classes previously explained.

The XMLTGUI class presents a GUI interface to the user through which he may type in or load from file a query to execute, execute the query over the network, and view the results of the query. The XMLTController class handles all interaction between the

XMLTGUI and XMLTDistributedClient in order to keep the two classes decoupled from each other.

# CHAPTER IV

# RELATED WORK

## Crossing the Structure Chasm [1]

In this paper, the authors observe that most of the world's data exists outside of database systems. The primary reason they give for this is that the majority of data outside of database systems is unstructured. From a user's point of view, unstructured data has several appealing properties, such as ease of authoring, querying and data sharing. In contrast, dealing with structured data, such as that found in a database system, is more complex, even though it offers its own set of advantages, such as richer query languages and more precise answers to queries. They call this difference between the world of structured data and the world of unstructured data the "structure chasm" and claim that by attempting to bridge it they hope to leverage the advantages of both worlds. Along this line they introduce the REVERE System, a development on their part which includes three innovations [1]:

(1) A data creation environment that entices people to structure data and enables them to do it rapidly;
(2) A data sharing environment, based on a peer data management system, in which a web of data is created by establishing local mappings between schemas, and query answering is done over the transitive closure of these mappings;
(3) A novel set of tools that are based on computing statistics over corpora of schemata and structured data.

The idea is that a cooperating user will enter their (local) unstructured data into the system, and the system will structure the data in a meaningful, system-usable way.  This includes understanding and extrapolating key and other constraints imposed by the data. There would also be tools for establishing source-to-source schema mappings between the local source and other sources that have structurally similar schemas.

One of the components of the REVERE System which the authors describe is Piazza, a peer data management system (PDMS) that allows for decentralized, ad hoc management of data at the peer level; peers can act as data providers, mediators, or query nodes.  Semantic mappings are developed between small sets of peers.  Queries can be posed using the local schema of one peer and transitively traverse the system to obtain a complete answer.

## Efficient Query Reformulation in Peer Data Management Systems [5]

In this paper, Tatarinov and Halevy build upon some of the work they had done in "Crossing the Structure Chasm" [1].  They start with a view of PDMS's in terms of individual peers being associated with a schema that represents its local data.  Semantic mappings between pairs or sets of peers provide "semantic paths" by which a query issued at one peer will be reformulated and traverse the network obtaining relevant data from any reachable peer.  The authors describe techniques for optimizing the query reformulation process, focusing on ways to prune reformulation paths and minimize reformulated queries.  They also demonstrate the advantages of pre-computing the semantic paths within the PDMS.  Their research is tested out on (almost) real-life XML data sets using Piazza.

## Continuing XML Interoperability Research

Research and development on the Lakshmanan and Sadri approach to interoperability on XML data [3] is being continued by researchers under the direction of Dr. Sadri.  One such researcher, John F. Harney, examines issues of query optimization in his thesis [2].  As the original authors identified, processing global queries that require inter-source sub-queries can result in an exponentially growing number of joins which must be performed (see the Previous Work section).  In their paper, the original authors discuss these issues and propose techniques for identifying when inter-source processing must be done and how to more efficiently carry out that processing.  In his thesis, Harney expands upon these optimization approaches and proposes additional techniques.  As part of his research, he provides partial C++ implementations of the query translation algorithm and his query optimization algorithms.

# CHAPTER V

# CONCLUSIONS AND FUTURE WORK

In their paper, Laksmanan and Sadri introduce an approach to interoperability on XML data residing across a network of heterogeneous sources [3]. In the later work of these authors and other researchers, their concepts are further enriched with the introduction of new ideas and more detailed developments. This thesis has built upon the existing body of work under this approach to interoperability on XML data. By implementing a peer-to-peer network we move the system further away from reliance upon any one authoritative central component. The implementation of the network also provides future researchers with a platform on which they can further contribute to the system and conduct extensive testing.

Our exploration of XML merging provides another useful piece to the picture by creating a mechanism for the combination of query results. XML merging also opens up new possibilities in other areas of XML research. With the continuing rise of the popularity of XML, particularly in areas of data storage, application interaction, and configuration management, the merging of XML data is likely to become a more visible subject.

## Future Research

In the course of designing our merging algorithms, we uncovered two strong topics of potential future research. The first topic relates to determining how XML data is merged together. Our merging algorithms use a simplified version of DTD, and certainly the loosening of this restriction, particularly in terms of key constraints, would allow for more flexibility in determining when two XML elements should or should not be merged (beyond whether they logically can be merged). But in addition to this, there are several potential data sets where a DTD alone is not sufficient to determine when it is meaningful or not, in terms of the corresponding application domain, to merge multiple XML elements with the same values. Consider for example, XML documents that store statistical data; in this application domain, it may be desirable for the merge of two XML documents to contain multiple elements with the same values and though the corresponding DTD may indicate that multiple XML elements of the same type are permissible, the DTD cannot indicate whether XML elements with the same value should or should not be merged together. One potential solution to this would be to add custom markup tags to the DTD or otherwise capture application domain specific meta-information, which would be used during the merge. W3C's XML Schema [11] may at least partially accomplish this with its richer language.

The second topic of future interest we identified is related to the user generated global query. To merge XML data together we require a DTD against which the data is valid. However, when a user within the system generates an XQuery at one of the local sources, the user can specify an arbitrary (assumed XML) format and structure for the result, as they define it within the XQuery itself.  So the issue emerges of determining a corresponding DTD to a given XQuery.  Research and commercial products designed to generate a DTD from a given XML document exist, but to our knowledge this has not yet been leveraged into the area of XQuery.

Our system implementation assumes a simplified distributed network model in which all clients can connect directly to each other.  Current research in the field of peer-to-peer networks appears focused on more loosely connected and much more complex architectures.  Additional research and expansion of our network in this direction is of future interest, particularly as this may generate new possibilities for our system to leverage some of the related XML interoperability research being done which also utilizes peer networks or peer data management systems.  To this end, extensive performance testing on our system would almost certainly be valuable in determining the evolutionary course the network implementation should take.

## Future System Development

We have identified a few system enhancements for both the XMLMerge project and the XMLT project, which if implemented would substantially expand the functionality of the system in the direction of the overall goal of implementing the Lakshmanan and Sadri approach to interoperability on XML data [3].  As such we consider these enhancements to carry the highest priority in terms of the future development of the system.

For the XMLMerge project, the top priority is to extend the merging algorithms to handle more of the functionality of DTD as defined in theW3C Recommendation [10], or possibly modify the merging algorithms to work with the W3C's XML Schema [11]. This would allow the system to perform more flexible, meaningful, and useful merging as valuable mechanisms such as key constraints and uniqueness could be utilized.

For the XMLT project, a top priority is the implementation of the query translation algorithm. We are currently working to convert and enhance John Harney's partial (C++) implementation of the query translation algorithm into Java. This will allow the network peers to use their local semantic mappings to translate a global query into an appropriate local query, as envisioned in the overall system.

Another priority for XMLT is to implement a form of query coordination at each peer. In the original conceptual system, there was a single coordinator to handle (global) query optimizations and inter-source join calculations, as well as performing the joining itself. As this does not fit a peer network architecture, we envision implementing a coordinator module at each peer which would perform these optimizations and calculations for queries originated at that peer. These coordinators would need certain information about the data available at the various nodes on the network; the network could be enhanced such that peers are able to communicate this information amongst each other upon request. With the implementation of these coordinators, the previous research done on query and join optimization could be applied to the system.

# REFERENCES

[1] A. Halevy, O. Etzioni, A. Doan, Z. Ives, J Madhaven, L. McDowell, and I. Tatarinov. “Crossing the Structure Chasm”. In *Proceedings of Conference on Innovative Data Systems Research*, 2003.

[2] J. F. Harney. "An Optimization Approach for XML Data Integration". Master Thesis, The University of North Carolina at Greensboro, 2005.

[3] L. Lakshmanan and F. Sadri. "Interoperability on XML Data". In *International Semantic Web Conference*, 2003.

[4] A. Silberschatz, H. Korth, and S. Sudarshan. *Database System Concepts*, 4th edition. McGraw Hill, Boston, MA, 2002.

[5] I. Tatarinov, and A. Halevy. “Efficient Query Reformulation in Peer-Data Management Systems”. In *SIGMOD Conference*, 2004.

[6] http://jakarta.apache.org/oro/

[7] http://java.sun.com/

[8] http://www.saxonica.com/

[9] http://www.w3.org/TR/rdf-syntax-grammar/

[10] http://www.w3.org/TR/REC-xml/

[11] http://www.w3.org/TR/xmlschema11-1/

[12] http://www.w3.org/TR/xpath

[13] http://www.w3.org/TR/xquery/

# APPENDIX A – CLASS DIAGRAMS

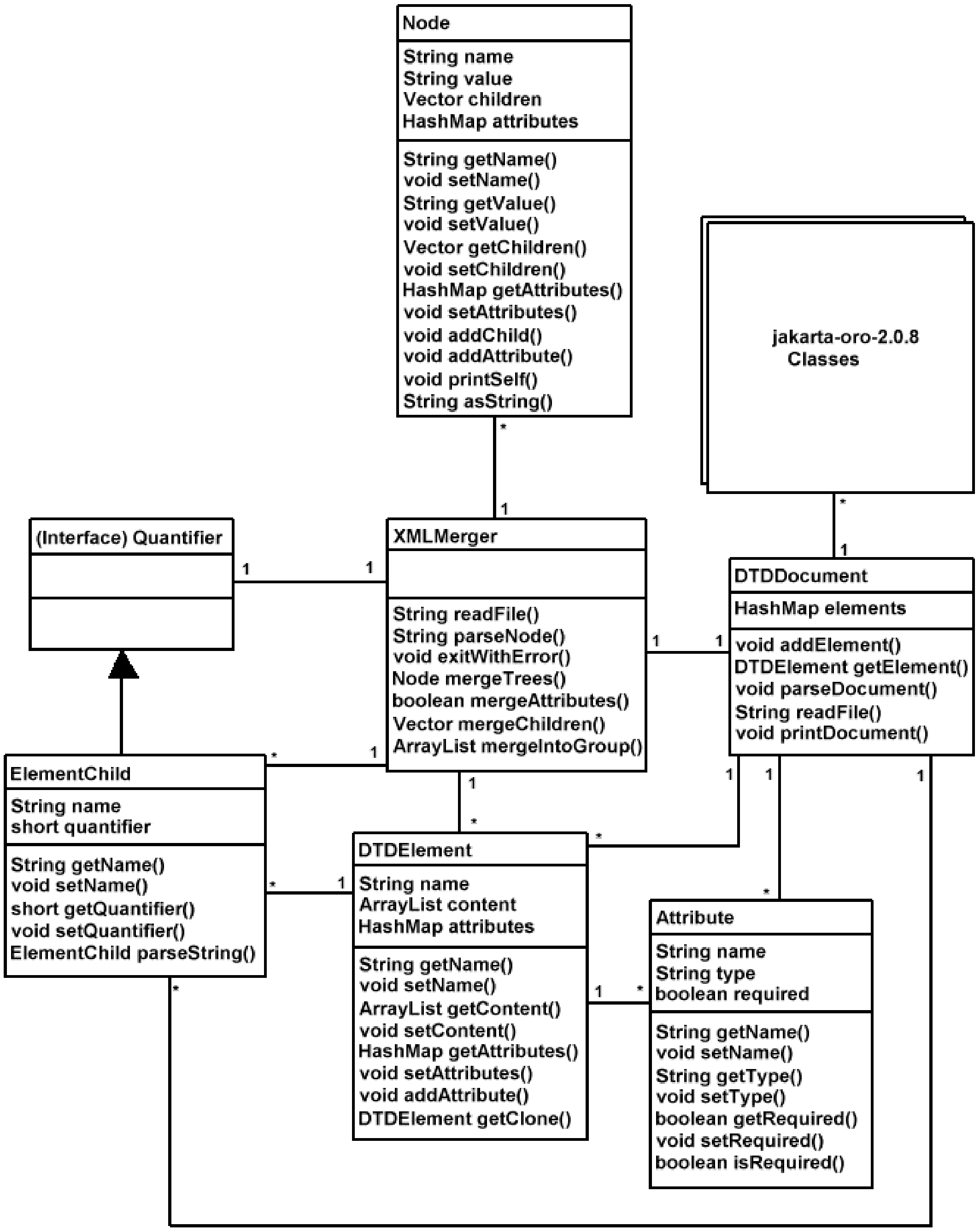


*Figure A.1: XMLMerge Class Diagram*

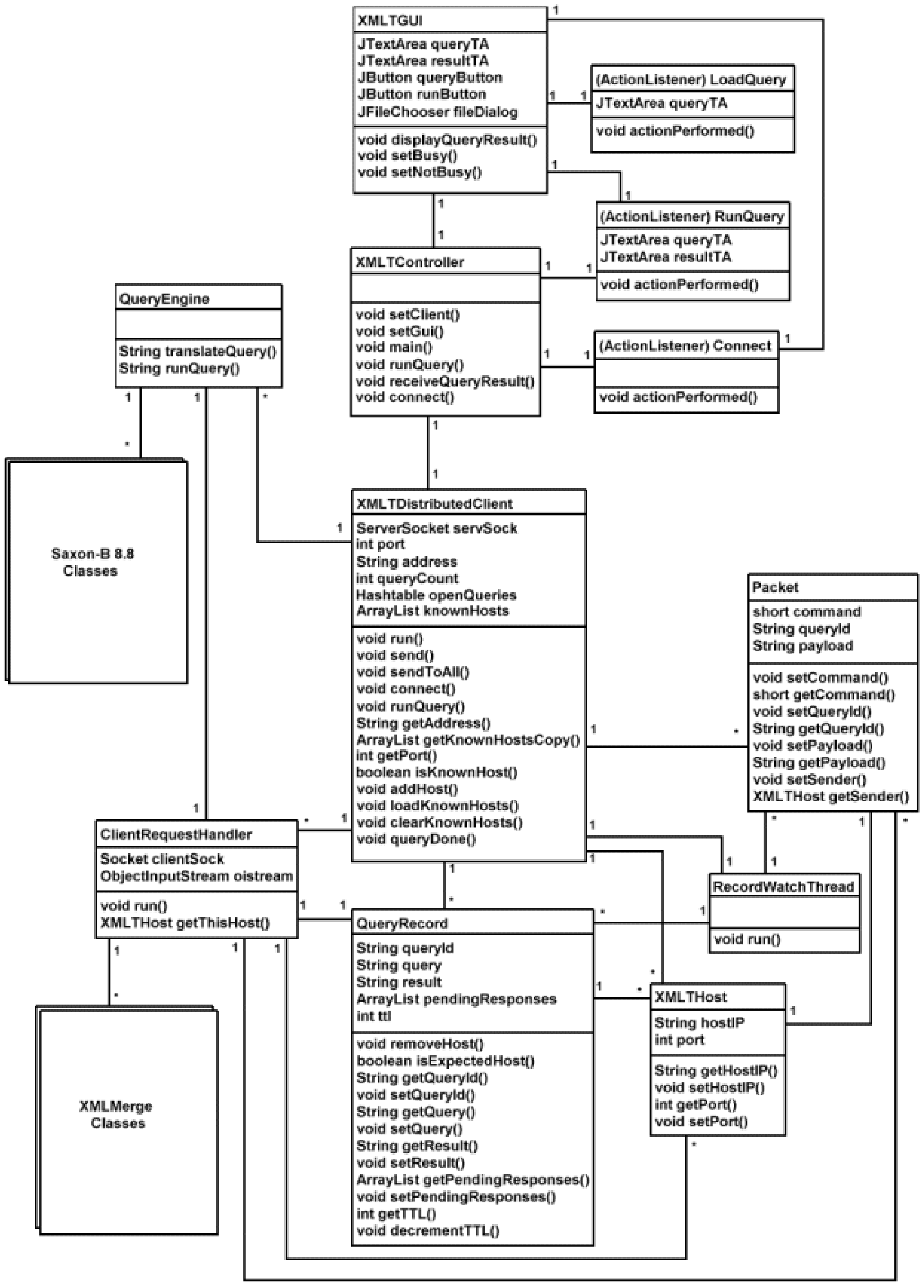


*Figure A.2: XMLT Class Diagram*