Query Languages and Tools for XML Documents and Databases

Sonali Sheth

John A. Miller

Computer Science Department

415 GSRC

University of Georgia

Athens, GA 30602-7404

jam@cs.uga.edu

Abstract

XML (eXtensible Markup Language) is fast being established as a standard for data exchange on the World-Wide Web. QT4XML is a query tool for querying information from XML documents. It uses a different approach from traditional query languages. It is a Visual Query Builder , with very little learning needed about any query language. The information to be provided by the user is largely in the form of XML grammar. So, any XML aware person will find this tool easy to use. In this paper, we review existing work in the same area and discuss the implementation of QT4XML. We also look at an application where QT4XML can be useful.

1 Introduction

The World-Wide Web is currently the largest resource of information, which is easily accessible. Untill recently it was difficult to express complex real-world structures and relationships by means of Web documents. The eXtensible Markup Language (XML) [Bray et al., 1998] is a new standard that supports data exchange on the World-Wide Web and provides a solution to the problem. Large collections of XML documents may be stored for retrieval using modern Database Management Systems (DBMSs). As XML becomes the dominant language for client side Web applications and documents, the number of computer users that will be XML literate is expected to become very large.

The Web data has a loose or undefined structure that is best described as "semistructured data". The database research community has developed the dynamically typed data model known as the semistructured data model [Abiteboul et al., 1996] [Buneman, 1997]. On the other hand, XML has been developed by the document community as a format that adds structure to documents, to standardize and ease the transmission of data via documents. These directions by the document and the database community are leading towards similar goals.

Before addressing the issue of choosing/developing query languages or tools suitable for accessing XML documents and databases, we briefly review the evolution of query languages starting in 1974 with SEQUEL and leading to today's SQL3 and OQLv.2 as well as emerging languages based on XML. Figure (1) depicts this evolution of query languages.

Relational databases started with the intention of providing an abstraction to the user such that the user would not have to be concerned with storage structure, rather would view data only as relations. Queries could be expressed in simpler higher level languages making way for rapid

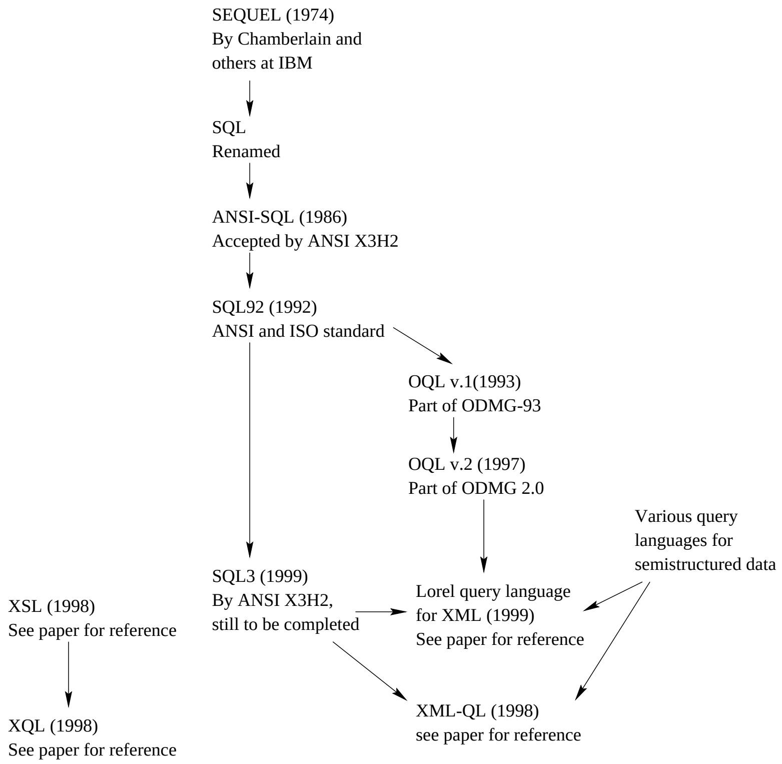

Figure 1: History of query languages

database development. As seen, SQL (Structured Query Language) [Ullman and Widom, 1997] for relational databases has matured starting with SEQUEL, then becoming an ANSI and ISO standard later as SQL92.

In the middle 1980s research was started on a different kind of database which could store classes and objects and their methods. This led to the development of the the object-oriented DBMS (OODBMS). The current OQL (Object Query Language, v.2) [Cattell et al., 1997] as we know today is a result of the ODMG (Object Database Management Group) [ODMG, 1998]. It was developed as a declarative language for querying and updating database objects. OQL also includes object extensions for object identity, complex objects, path expressions, operation invocation and inheritance.

With the founding of ODMG and the increasing prevalence of object-oriented programming languages, SQL3 for object- relational DBMS (ORDBMS) was thought of. The work is still going on and the target for completion is currently 1999.

As explained above, the current state-of-the-art in database management is represented by the Object-Oriented (OO) and Object-Relational (OR) DBMSs each with their own query languages, OQL v.2 and SQL3, respectively. Web documents (in this case XML documents) can be stored in these databases, but as they do not conform to either the relational or object-oriented model, storage is not a simple issue. Hence a few problems remain: (1) the OQL and SQL3 query languages are not ideal for accessing semistructured documents, (2) they lack sophisticated information retrieval capabilities and (3) they require substantial learning.

A solution to this problem can be found within XML itself. A DTD (Document Type Definition) defines the schema for an XML document. If the query is based on a DTD, as in QT4XML, it can be designed so that little learning of existing query languages is needed. Currently, there are a few popular query languages for XML viz. XML-QL [Deutsch et al., 1998] with SELECT-WHERE construct like SQL and features of query languages developed for semistructured data, Lorel [Abiteboul et al., 1996] which is an extension to OQL and XQL [Robie et al., 1998] which is based on the XSL (Extensible Stylesheet Language) [Deach, 1999].

In this paper we discuss the basis for our QT4XML (Query Tool For XML) and its high level implementation. It surveys the current work being done on query languages and tools for XML, and then discusses our implementation of a simple XML query tool which can be used for building queries in a visual manner, and can be used to query against any database which supports storage of XML documents in some form (tree like graph, Objects, etc). The query tool also provides for creation of Java classes which can be used for storage of XML documents as Java objects in the database, as in our prototype implementation.

Section (2) discusses the use of XML as a universal format for data interchange. Section (3) explains the structure of Web information and problems associated with current query languages trying to query Web information. In Section (4) we discuss the different modules of QT4XML and highlight its features by means of query examples and screen shots. Section (5) gives an analytical comparison of some of the key features of existing query languages. Finally in Section (6) we discuss the scope for future development.

2 Universal Data Exchange Language

Modern information systems consist of many applications and subsystems that must work together to support the information needs of organizations. In the past, it has been very difficult to get separately developed software to interoperate. Fortunately, recently developed applications and subsystems have one thing in common, they all utilize objects (e.g., OO Programming Languages such as Java and C++, OO and OR Databases, and middleware such as CORBA and DCOM). Unfortunately, the form of object varies. Rather than mapping an object to all other possible forms, it would be better to have one form as the *standard for data interchange*. It appears that several companies believe that XML will be that standard.

All core applications based on simple, structured ASCII text – e-mail, Usenet news, the Web, rely on fixed data-exchange formats. This requires that a new exchange format be specified each time a new application is used. Data integration solutions may use XML to automate the exchange of data. In general, with an integration solution, XML serves as an interface layer or wrapper for data being passed between data sources, making it possible for a wide variety of applications, legacy systems, and databases to exchange information. XML can thus be said to be Web-style Electronic Data Interchange (EDI) standard.

Let us look at some of the possibilities for data exchange with XML as the common format. Figure (2) depicts XML as the key ingredient through which disparate data systems interact.

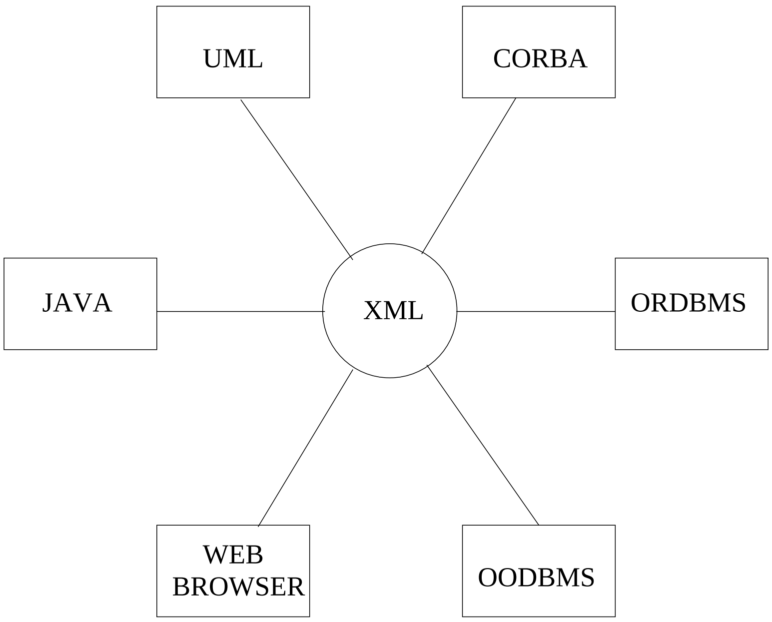

Figure 2: XML as a Universal Data Object

- **Java**. Java and XML are very complementary. Java can dynamically produce XML documents. XML forms may send data to Java Servlets much as HTML forms now send data to CGI scripts. To facilitate the storage of Java objects in a form that other tools and systems can understand, Java objects in main memory can be made persistent as XML documents (or parts of an XML document) rather than as binary files as is currently done when Java objects are serialized. The coins project has a prototype system to do just this [Laforge, 1998].

- **Web Browsers**. An XML document should be thought of as data first and as a displayable document like an HTML page second. The display rendering information is given separately as an (eXtensible Style sheet Language) XSL [Deach, 1999] specification. Currently, there are a few browsers that will display XML documents, but many more are expected to become available in the near future. Available browsers include Jumbo [Murray-Rust, 1998], Amaya [Vatton et al., 1999], Fujitsu HyBrick [Fujitsu, 1999], Netscape Communicator 5 source code release [Alschuler and Walter, 1998], and Internet Explorer 5 release [Turner and Friedman, 1999].

- **Object-Relational Database Management Systems (ORDBMSs)**. Since XML documents may exhibit complex structures including deep nesting, traditional relational databases are not ideally suited for storing such documents. Object-relational databases augment relational databases with object-oriented capabilities making them more suitable for storing XML documents. Most object-relational database systems currently implement a subset of the ANSI/ISO draft SQL3 [Ullman and Widom, 1997] standard. *Oracle 8i* is Oracle's object-relational database for internet computing and it supports information definition and access through XML. Oracle sees XML as the means of providing a standard definition of all enterprise resources in one place, regardless of the type and location of the information. The Oracle XML Parser can be used for programmatic processing of XML documents or document fragments. Oracle8i uses the structures in XML to build database structures and store data or links to data (through locators) according to the rules and constraints defined in XML. It automates parsing and rendering of data between XML and the database. Individual XML instance objects are decomposed and stored according to the defined structure. It provides for XML-enabled "section searching" for more precise searches over structured documents. Upon access, the data is pulled and the XML object reconstructed and presented through the browser. XML provides the promise of establishing a coherent network of all kinds of information content not only within a system, but across systems. Oracle8i's support of XML means that this powerful capability will be supported, and its integrity ensured, by the Oracle DBMS [Oracle, 1998].

- **Object-Oriented Database Management Systems (OODBMSs)**. Object-oriented databases are designed to handle objects in their native forms. The objects then maintain their own data, methods, relationships and semantics of the whole model. This is ideal for the creation and management of hierarchical XML trees, while providing both hierarchical tree navigation and rich link traversal. XML also complements object-oriented databases in that, data from such a database, does not need to undergo transformation to be used at the client side; XML handles all of that. A couple of object-oriented databases that provide early support for XML are Object Store [ODI, 1998] and Poet [Poet, 1997].

Object Store, can store a tree-like structures as-is, eliminating the mapping code required by traditional relational databases to convert the hierarchically structured XML object into rows and columns. With Object Store, navigating an XML tree is as simple as traversing a pointer in local memory. Object Store also has index and query capabilities which facilitate high-speed queries. It can support a data abstraction model for XML, allowing developers to add a new field or attribute to a data structure without having to redefine the object model or evolve the schema. Object Store with its ability to manage, serve, index and query native XML data, thus provides a good foundation for many data integration solutions. We discuss Poet in detail in our implementation section.

- **Common Object Request Broker Architecture (CORBA)**. CORBA is a major standard for distributed object computing. Most CORBA products provide mechanism for making CORBA objects persistent. For maximal interoperability, the states of CORBA objects may be saved as XML objects in documents. CORBA's Interface Definition Language (IDL) defines the interface to objects. The IDL class members can be converted to corresponding elements in an XML DTD (Document Type Definition). Also different data types can be mapped to data types in XML with some modifications.

- **Unified Modeling Language (UML)**. UML is positioned as a general-purpose, unified, visual modeling language for software development, in particular for object analysis and design. Its main components are a metamodel, and a notation for graphically representing the concepts from the metamodel. It provides one language to the software and system development community, ending the method wars of the past few years at least in the area of concepts used to model object-based systems [MetaModel.com, 1999]. With XML gaining momentum throughout the industry, recently several efforts have been started to integrate existing metamodels and metamodeling with XML. Design tools based on UML can generate code or specifications in several languages and formats. Generating XML DTDs from a graphical design can be done much like database schema are generated today. Some work is being carried out by the CDIF Technical Committee [Ernst, 1997]. CDIF is a Family of Standards that lays out a single architecture for exchanging information between modeling tools, and between repositories, and defines the interfaces of the components to implement this architecture. The XML-based CDIF syntax allows the exchange of meta-models and models using the emerging XML standard.

3 Semistructured Nature of Web Information

With the Web presenting a vast source of information, it is seen that there has been a growing need and interest in managing data that the Web represents. Web documents are constantly evolving and so do not have a fixed schema which can define them. The data that the Web presents can be described as semistructured, because it can have both structured and unstructured parts. Semistructured data implies data that does not have a rigid or a complete structure, e.g. data which might have a schema, but which places only loose constraints on it.

The issue of semistructured data is becoming important as it arises not only when we consider the Web, but also during integration of heterogeneous sources of data, in a variety of non-traditional applications of databases and information systems, including the databases in molecular biology, ACeDB genome database [Thierry-Mieg and Durbin, 1992] or management systems for Web sites. Data in these applications have irregular structure: some elements have missing components, others may have multiple occurrences of the same component, different types may be used to represent the same kind of information, etc. Hence, providing for a database to store such information is quite difficult.

Having seen the nature of Web information or documents, we look at the drawbacks of the currently available query languages: (1) the object-oriented and object-relational query languages are not ideal for accessing semistructured documents. These languages are designed for querying fixed and defined schema, whereas for semistructured data, the schema is not rigid. (2) they lack sophisticated information retrieval capabilities, like querying to arbitrary depths, and (3) they require substantial learning. So now, there are two issues involved, one of a query language to access semistructured data and other of a database to store it. There are a few solutions available at this time, which tackle one or both of the above issues. However, a close look at them reveals much left to be done.

Regarding the issue of defining a query language, it is seen that with not many databases having provided direct support for XML so far the focus is on OQL with suitable extensions to act as a XML query language. A query language for XML should provide expressive and easy access to data. It should also be extensible to accommodate for extensions in XML.

The currently popular databases that can be used to store semistructured data (in our case XML) in some form are basically object-oriented databases that allow storage of Java or C++ objects formed from XML documents [Poet, 1997] [ODI, 1998]. So, there is a need for mapping between XML documents and the corresponding objects that the database supports and is hence not convenient. There has been a move in the right direction with eXcelon [ODI, 1999], which allows for storage of XML documents as is, with many features for managing and querying data.

We shall look at QT4XML and its Visual Query Builder [EECS, 1999] [Aeronaut, 1996] features with the help of examples. We shall also discuss a few of the promising query languages in the following sections.

4 Prototype Implementation of QT4XML

In our project, we develop a visual query tool (QT4XML) for XML, which is easy to use and can be developed as a complete query language in the long run. The prototype also provides for creation of Java classes which can be used for storage of XML documents as Java objects in the database, as in our prototype implementation.

In the following section, we take a look at the underlying concept of QT4XML and how complex queries can be easily expressed using the Tool. In the next section we take a peek at the internals of the system of which the Visual Query Builder is an important module. QT4XML does not have a complete grammar like the existing query languages for XML, as it tries to represent what it can visually and the rest is done syntactically.

4.1 Converting XML to Database Schema

If XML documents are required to have a Document Type Definition (DTD), they are called valid XML documents. (When no DTD is given, the document is simply said to be well-formed.) It is straightforward to convert a DTD into an object-oriented schema specification. For example, the following DTD can be readily converted to OQL's Object Definition Language (ODL).

Consider XML DTDs for office MEMO Figure (3) and MEETING (4) document types, adapted from an XML tutorial [Harold, 1998].

```
<!DOCTYPE MEMO [
    < !ELEMENT MEMO (TO, FROM, SUBJECT, BODY) >
    < !ATTLIST MEMO MEMO_NUM ID #REQUIRED >
    < !ATTLIST MEMO MEETING_REFS IDREFS #IMPLIED >
    < !ELEMENT TO           (#PCDATA) >
    < !ELEMENT FROM         (#PCDATA) >
    < !ELEMENT SUBJECT      (#PCDATA) >
    < !ELEMENT BODY         (P+) >
    < !ELEMENT P            (#PCDATA) >
] >
```

Figure 3: Memo Document Type

```
<!DOCTYPE MEETING [
    < !ELEMENT MEETING (ATTENDEE, PLACE, CALLEDBY) >
    < !ATTLIST MEETING MEETING_NUM ID #REQUIRED >
    < !ATTLIST MEETING MEMO_REFS IDREFS #IMPLIED >
    < !ELEMENT ATTENDEE     (#PCDATA) >
    < !ELEMENT PLACE        (#PCDATA) >
    < !ELEMENT CALLEDBY     (#PCDATA) >
] >
```

Figure 4: Meeting Document Type

The MEMO and MEETING DTDs, each contain an ID type attribute to uniquely identify each memo/meeting. The attribute MEMO_REFS in MEETING is the list of memos discussed in that meeting while the attribute MEETING_REFS in MEMO is the list of meetings in which the memo was discussed. Expressed in ODL, the DTD for MEMO is shown in Figure (5).

Thus, each element or attribute in a DTD can be mapped to an element in the object-oriented schema. For complex DTDs, for example, a DTD in which the subelements of an element may have subelements, the translation is a bit tedious, but is still possible.

4.2 Prototype Architecture

Figure (6) shows the architecture of the entire system. Before we explore the different modules in Figure (6), let us take a look at our choice of database (POET).

```
interface MEMO
      (extent MEMOS
      key (FROM, SUBJECT))
{
      attribute set             TO;
      attribute set             FROM;
      attribute set             SUBJECT;
      attribute set<string> BODY;
}
```

Figure 5: DTD of Figure (3) as Expressed in ODL

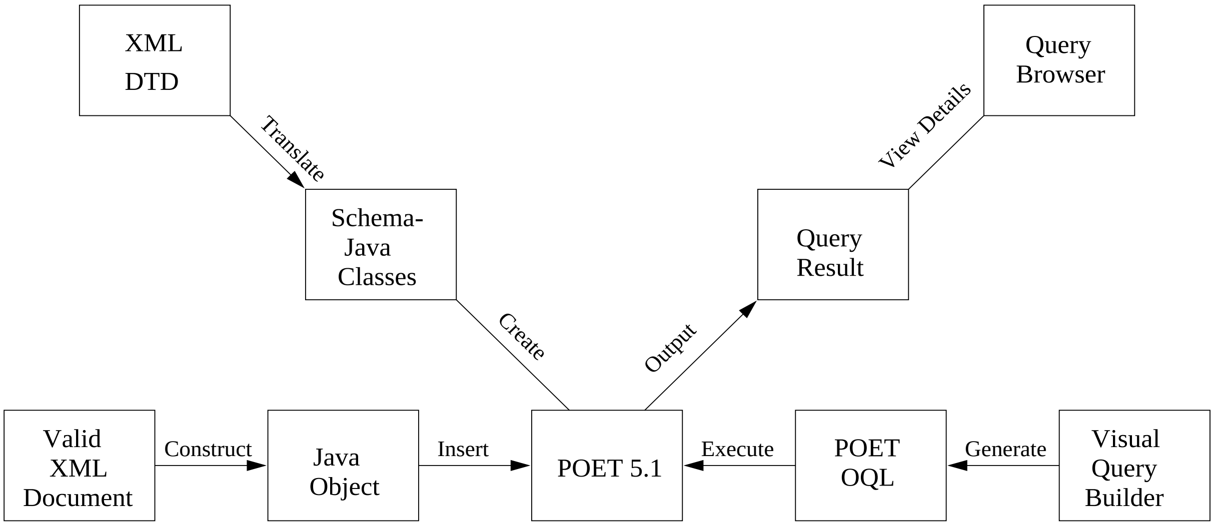

Figure 6: Project Design

4.2.1 Database

When using Java, it is possible to use its features like serialization to store data, i.e., to write objects to disk. Although this presents a simple way to make data persistent, this is possible only for a small number of objects. Also, essential features like transactions, recovery and reorganization utilities are not available with such an implementation. *Poet Object Server* addresses the needs of XML as well as the needs of associated applications. The choice of database is specific to this implementation. It can very well be replaced by any other database, the only change being that the query language to which the visual query is translated to, will then be the language that the new database supports. Poet provides an object-oriented database that can be scaled up as well as down while leveraging the same APIs, thus simplifying application development. Also it is flexible, reliable and provides features like multi-user/concurrent access, queries, user authorization, etc. The POET Java SDK allows any Java program to store objects in a POET database, with data represented in memory as true Java Objects. The same POET database can be accessed by any other application. The database can be copied across machines, across different operating systems. Thus the database is compatible across all platforms and programming languages. It provides a language-independent object model which allows developers to mix and match C++ and Java even in a single application. It also offers simple standards-based APIs and embeddability. It also provides for a higher granularity locking, down to element level, which is critical for user scalability. In addition, Poet provides features like versioning, management of arbitrary links, import/export, publishing of structured content on the Web and support of object-oriented programming languages.

There are a few features that POET is unable to support in its current version, but if provided would would make the database easy to use for complex queries, etc. We list some of the features desired:

- support of complete OQL language. (this would enable the user to exploit all the features of OQL),
- projection on elements of a collection type,
- multiple projections,
- ability to specify selection criteria on elements of different types from a collection of objects,
- selection from multiple objects (in our case documents) and
- an easy way to express structure of recursive DTDs.

4.2.2 DTD to Java Class

As we have seen in the discussion earlier, the most popular databases which provide data storage almost as rich as that required for XML are OODBMSs. So in our implementation, we use POET, an Object Database which supports storage of Java Objects. QT4XML parses any valid DTD using IBM XML Parser for Java [IBM, 1999] and creates Java Classes corresponding to each element declared in the DTD. These classes are used to instantiate objects which are populated using data from the XML documents, as explained in the next section. All these classes are also registered with POET, which is analogous to defining a schema in a traditional DBMS.

4.2.3 XML to Java Object

QT4XML also parses any XML document, using IBM XML Parser for Java, and instantiates objects at run time using the Java classes described earlier (of the corresponding DTD). These objects are inserted into the POET database. This module requires the DTD to be available along with the documents (a valid document is needed).

4.2.4 Visual Query Builder

This is the heart of the system and provides user friendly features for constructing queries. This is explained in detail in the next section by means of expressing various queries with the help of the Visual Query Builder. The chief feature of this Visual Query Builder being that the user need not be aware of the existing query languages like SQL3 or OQL v.2, nor does he need to learn any query language for forming queries using QT4XML. The user just needs to know the overall structure of any DTD/XML document and it is just a few clicks to form the query.

4.2.5 Fetching query results

Our prototype can be used either in a client-server mode, wherein, the query formed using the Visual Query Builder (client) is communicated to the server and the results after execution are fetched back or in a standalone mode in which the query built in POET OQL form is stored in a file at the client side. The standalone mode is useful in case of server being inaccessible. The modes can be switched by minor changes as explained in the documentation of the PoetGenerateCode class.

- **Standalone QT4XML.** The query is designed using the Visual Query Builder as explained in Section (4.3). When the "Generate" button (see Figure (7)) is pressed, the query is translated into POET OQL format and is stored in a file. This query can be executed using the POET Developer's Workbench.

- **Client-Server QT4XML.** As above, the query is generated, when the "Generate" button is pressed. However, now, instead of saving it to a file, QT4XML uses Java RMI to communicate the query to the RMI server. The RMI client calls a method **executeTask(***query***)** to pass the query to the server. The server connects to the database which contains the XML documents and executes the query on the database, using POET API. The results are fetched and returned to the RMI client and displayed in a new frame.

4.3 Visual Query Builder

As in the earlier section, a DTD can be used to define a schema. We use this schema definition for querying XML documents (which correspond to that DTD). Patterns may be substituted into a DTD to form the basis for a query. Such a query can be run against one or more databases, in which case all matching XML documents will be retrieved for which the conditions in the query are true. This form of querying a database is analogous to the approach used in Query-By-Example (QBE) [Zloof, 1975].

The Visual Query Builder is used for defining queries which can be translated to OQL or some such existing query language. The Visual Query Builder uses the DTD as the basis to form queries. Consider the query builder screen during query building as in Figure (7).

The user selects and places a query box on the canvas for each DTD involved in the query. "Updating" a query box brings up a dialog box in which the name of a DTD is given. QT4XML associates each query box (node) with a unique number, by using the name of the DTD associated with the node and a uniquely generated number as seen in the screen shots. Pressing the get button will place the DTD in the dialog box in the form of text boxes, labels, check boxes. For each DTD (query box), the text boxes may be modified to constrain the elements. The check boxes are provided for each element and checking an element, selects its value to be displayed in the output of the query result.

If the query involves more than one DTD, then accordingly the query boxes are created and associated with the corresponding DTD. Any join condition between queries is shown by a "link" between the query boxes. "Updating" the links, brings up a join box which allows the user to enter the join condition between queries. We shall describe more features as we come across suitable examples to illustrate the same.

4.3.1 Pattern Matching in XML Documents

Pattern matching in XML documents is a straightforward process. Consider the memo document type as shown in Figure (3). Suppose that one wishes to obtain all documents of type MEMO which are from John Miller and only see the "subject" of the memo. Instead of explicitly writing a where clause with a condition, as one would do in either SQL or OQL (select...from...where...), the DTD itself can be used to specify the where condition, using the Visual Query Builder as in Figure (7). As seen in Figure (7), the "root element window" contains the DOCTYPE element as in any DTD and the "element window" contains the subelements of the DOCTYPE element or any other subelement. The attribute or element you wish to display can be selected by checking the check box.

Here, we need to mention that several alternatives for the visual representation of elements and their attributes were considered before deciding on the current one as in Figure (7). One alternative being, only elements or only attributes be shown in any one window. This had the ease of understanding the DTD structure, but had the overhead of extra clicks necessary to build even a simple query. One other choice thought of was a graphical tree like representation of a DTD and ability to draw links to represent join conditions and pop-up windows to write constraints. However, the current alternative seemed ideal in terms of design, user friendliness and development effort and so was made the final one.

The same query may be written in OQL or SQL.

```
select m.SUBJECT
from   MEMO m
where  m.FROM = "John Miller";
```

Although simple in this case, complex queries may be hard to express in this form. To simplify a user's interaction the graphical query tool relieves the user from knowing accurate syntax and such issues. Our approach is to make writing a query as simple as modification of a DTD.

4.3.2 Pattern Matching using Regular Expressions

Simply finding XML documents that match constant strings is too crude of an approach to serve as a query language. XML includes several regular expressions operators that can be very useful in queries.

- | – alternation (match any of the cases)
- * – closure (repeat 0 or more times)
- + – closure (repeat 1 or more times)

The query in Figure (7) may miss certain memos since in some cases the middle initial may have been used. To handle both cases an alternation may be used. This is accomplished by changing the FROM element specification to the following:

```
FROM  == "John Miller" : "John A. Miller"
```

If one is interested in memos from any Miller, the FROM of Figure (7) is changed as shown below:

```
FROM  == ^*Miller
```

Suppose one is interested in memos which satisfy one of the following conditions

- memo is from Miller and to Sheth or
- the subject is "XML" followed by any other text or the subject is simply "DTD".

The query in Figure (8) introduces multiple conditions to constrain the values that the elements FROM, TO and SUBJECT can take on. It also shows the use of the AND and OR connectors to express boolean conditions between elements or attributes.

The conditions on elements are ANDed together. We use the conjunctive/disjunctive normal forms. So, if the AND button on the window is selected, we are using the conjunctive normal form and whichever elements are linked together are ANDed. Elements not linked are ORed together. If the OR button is on then it is vice versa as we are using the disjunctive normal form.

The Visual Query Builder shows how easy it is to depict OR and AND conditions in a query. The wild card character "^" and the regular expression operator "*" are used in conjunction to denote "any text" that follows the word "XML". We use the operator ":" to denote the OR condition in the Subject constraint. All the operators ":" (OR), "&" (AND), "|" (XOR), "-" (set difference) as in the grammar defined in the appendix are set operators. The constraint in the Subject field produces a set of values with which the XML data in the database is compared with. The OR and AND operators are used as boolean operators when we use the AND/OR connectors.

The ease of using the Visual Query Builder is illustrated in the next section by means of a complex query.

4.3.3 Complex Query

Consider expressing the following complex query using the graphical XML query tool as shown in Figure (9).

Query: Find the memos where the person who received the memo is the person who called the meeting for discussing the memo and whose name ends with "Miller".

Although the above is not implemented at this stage, because of limitations of POET DBMS, and other limitations, it is seen that QT4XML can be used to build queries that refer to different

DTDs, i.e., query on XML documents referring to different DTDs. The join condition between the DTDs can be specified in the dialog window for the link between the queries.

4.3.4 Recursive DTD Query

If a DTD has a recursive structure, it is possible to query to the desired depth using QT4XML. All the user has to do is to specify the condition at the desired depth on the Visual Query Builder, which only involves clicking of a few buttons. However, this does have the drawback of needing to pop up several windows if the depth of the query is large. We hope to extend/modify this feature in a way that this is avoided.

4.4 Joining/Linking Documents

ID attributes are used to provide a unique identity to an element and IDREF attribute can be used to refer to these IDs in XML terminology. The QT4XML allows for joins between DTDs, however it would be interesting to use IDREF(s) of DTDs for the same. Also there is this need of specifying links between elements of DTD through some feature advanced than the currently available features. What is needed is something analogous to a foreign key-primary key relationship as in any Relational DBMS, not just in DTDs, but across DTDs. XLink/XPointer provides capability to link across documents or to link across the elements of documents. But it is a textual way of representing links. Something similar to an ID is needed along with this linking mechanism to provide a join feature across documents. Currently some work is going on in a parallel area of correlating information across ontologies, asset types and resources on a resource-independent level by means of the Metadata Reference Link (MREF) [Bertram, 1998]. MREF enables producing of semantically related results by correlating semantic information of entities involved. Current search engines fall short in this regard, because they follow a textual search instead of a truly semantic one.

5 Comparison of Query Languages for XML

In the following subsections, we shall look at how each query language treats some important features and also the special or unique features of each query language.

5.1 XML-QL

XML-QL [Deutsch et al., 1998] is a query language submission to the World Wide Web Consortium (W3C) by the University of Pennsylvania, AT&T Labs, INRIA (in France) and University of Washington. The data model proposed for XML-QL is a variation of the semistructured data model [Buneman, 1997]. XML-QL has a SELECT-WHERE construct, like SQL, making it easy for users familiar with SQL. It also uses XML like tags to specify constraints or conditions. Let us see how the query from Section (4.3.1), "Retrieve the subject from all documents of type memo which are from John Miller" can be written in XML-QL. There are different ways to write the same query.

```
WHERE <MEMO>
        <FROM>John Miller</>
        <SUBJECT> $s </>
      </> IN "www.a.b.c/bib.xml"
CONSTRUCT <result>
            <SUBJECT> $s </>
          </>
```

We assume that our data source is the URI "www.a.b.c/bib.xml".

Some of the features of XML-QL are:

- XML-QL has a SELECT-WHERE construct like SQL and borrows features from query languages for semistructured data.
- The XML data model used is a variation of the semistructured data model [Buneman, 1997]. Every node in this XML graph has a unique object identifier (OID), which is the ID attribute associated with the corresponding element or is a generated OID, if no ID attribute exists for that element. IDREF attributes are also expressed in this graph which can be used for writing join expressions.
- Transforming and integrating XML data from multiple XML sources: In XML-QL it is possible to query several sources simultaneously and produce an integrated view of their data. XML-QL queries are structured so that they have blocks and subblocks. Their semantics is related to that of semi-joins in relational databases and so are a versatile and powerful feature.

OIDs (Object Identifiers) are used to help produce the join result. Skolem functions are also used to help transform or restructure results of a query. A skolem function in general is used to generate a new identifier for every distinct value of the result. Each block produces its results with each result having a unique OID based on some key field, using skolem functions. When a block's results are produced, they are joined with the earlier block results if the OIDs are the same. XML-QL currently supports only a limited form of semi-joins.

- The join conditions as explained above are implicitly ANDed together. There is no OR operator. The way do carry out OR conditions as explained by Alin Deutsch is: "One way of evaluating disjunctive queries is by writing them in disjunctive normal form and evaluating a separate query block for each conjunctive query, fusing nodes using Skolem functions. In addition to this, hidden disjunctions can be evaluated by external functions."

- XML-QL provides regular path expressions which allows querying of elements to an arbitrary depth in the XML-QL data model. The operators that the regular path expressions provide are the alternation (|), concatenation (.) and Kleene-star (*) which are similar to those used in regular expressions.

- There is no provision for specifying range queries directly as in some of the other query languages.

- Ordering: XML-QL supports two data models, an ordered one an unordered one. An ordered XML graph is like an unordered one, except that for each node, there is a total order on its successors. An ordered model has the drawbacks of complex semantics of the query language. Even in an unordered model, an ordering needs to be chosen when an unordered XML graph is transformed into a document, in order for it to conform say to a DTD. Ordering in a query can be specified by means of an order-by clause.

Following are some special features

- Translating data between different ontologies, meaning that we can write queries that extracts data from the database for one DTD and transform it into elements conforming to another DTD.

5.2 XQL

XQL (XML Query Language) [Robie et al., 1998] is a notation for addressing and filtering the elements and text (information) of XML documents. This is a joint proposal made by Texcel Inc, webMethods Inc and Microsoft Corporation. It is an extension to the XSL pattern syntax. It does not indicate the format of the output, rather just specifies the logical returns. The result of the query could be any structure like a node, a list of nodes, an XML document, etc. XQL builds on the XSL capabilities of identifying classes of nodes, by adding Boolean logic, filters, indexing into collections of nodes, etc.

As in Section (5.1), we write the following query in XQL "Retrieve the subject from all documents of type memo which are from John Miller". Again, there are different ways to write the same query.

```
MEMO/SUBJECT [MEMO/FROM/. == 'John Miller']
```

Some of the features of XQL are:

- XQL is a natural extension to the XSL pattern language. It is a concise language. The basic SQL syntax is like navigating a file structure.

  ```
  Memo/From
  ```

  In XQL this means finding the collection of From elements within Memo elements.

- XQL does not provide for join conditions across XML documents corresponding to different DTDs, though it does provide for operators as and, or, and not for querying within documents corresponding to a single DTD.

- XQL does not use any regular expression operator, but uses the '*' operator as a wild card character to specify any arbitrary path in the query.

- Provision of a subscript operator which allows a range of elements to be returned. For this, an expression is specified rather than a single value inside the subscript operator.

  ```
  author[0 $to$ 3]
  ```

  This XQL query finds the first through third author elements.

- The result of an XQL expression preserves document ordering, hierarchy and identity, if they are defined, i.e., a collection of elements is returned in document order without repeats and a collection of attributes is returned without repeats, but there is no implicit ordering because attributes are by definition unordered.

  Special features

- The specified conditions in the XQL query may be evaluated at any level of a document, without expecting to navigate from the root. XQL specification does not indicate the output format. Thus the result of a query can be anything from a node, or a list of nodes, an XML document or just any other structure.

5.3 Lorel

The Lorel [Abiteboul et al., 1996][Goldman et al., 1999] language was originally designed for querying semistructured data. Lorel has been extended and modified to be able to query XML data efficiently. It is an extension to OQL, with a SELECT-FROM-WHERE structure. It relieves the user from the strict typing of OQL and provides powerful path expression.

As in Section (5.1), we write the following query in Lorel "Retrieve the subject from all documents of type memo which are from John Miller". Again, there are different ways to write the same query.

```
Select MEMO.SUBJECT
From MEMO
Where MEMO.FROM.Text = "John Miller"
```

Some of the features of the Lorel query language are:

- Lorel query language is an extension to OQL with the SQL/OQL style of querying.

- In Lorel, an XML data model can be represented either as a literal tree, where the IDREF(s) attributes are just text strings or as a semantic graph where the IDREF(s) attributes are actual links. Lorel supports both these modes allowing an application to choose between them. This is important when we consider the fact that the application may or may not be interested in the information presented by IDREF(s) attributes.

- It provides some language constructs to transform data and return structured results. One of the constructs is the "with" clause used with the select-from-where form. It causes the query result to replicate all data selected by the select clause along with all data reachable via path expressions in the with clause. The query language also supports Skolem functions for expressive data restructuring.

- It does support joins on XML documents from different sources, but not much information is available on it currently.

- Path expressions can include wild cards or regular expression operators. These together provide a great deal of flexibility to the query language. See table below for details.

- Range qualifiers can be added to path expression components or variables. When this is added, the matched values are limited only to the range specified in the range qualifier. The range can be a list of single numbers and/or ranges, e.g., [1-3, 5].

- Provision of an Order-By clause: The result of a query is an ordered list of identifiers of the elements that match the query criteria. Lorel provides an "order-by- document-order" order-by clause amongst others, so that the query result is ordered based on the original XML document. If there is no order-by clause in the query, the ordering is unspecified.

Special features

- Support of a declarative, expressive update language, unlike most other semistructured and object-oriented languages.

- Use of DataGuides when a DTD is not supplied, that provide concise and accurate structural semantics of the underlying database.

5.4 Analysis

Lorel is defined as an extension to the ODMG model and the query language as an extension to OQL. This is important considering the fact that the ODMG model comes closest to a semistructured data model. Also unique about this language is that XML links can be used in different ways, viz. literal and semantic as explained earlier.

XML-QL and Lorel provide for data transformation primitives for selection, extraction of XML data. XML-QL also has an ordered model to preserve document ordering and to facilitate querying

and data manipulation. Lorel has ordering primitives to output query result in the document order and other orderings. XQL does not specify the output format of a query.
XQL is the most compact query language for XML amongst XQL, XML-QL and Lorel.

Lorel is the only language amongst the three discussed, which has an update query language defined.

None of the three languages support XML Namespaces currently, though XQL has identified its requirements and XML-QL proposes to support it in the future.

Let us compare the 3 query languages for XML discussed above with the underlying query language for QT4XML.

Support for	**XML-QL**	**XQL**	**Lorel**	**QT4XML query language**
Comparison operators	<, <=, >, >=, =, ! =	=, ! =, <, <=, >, >=	<, <=, =, <>, >=, >, like, grep, soundex	==, ! =, <, >, <=, >=, <ALL, >ALL, <=ALL, >=ALL
Boolean operators	Implicit AND, see Section (5.1)	\$not\$, \$and\$, \$or\$	not, and, or	Refer Section (4.3.2)
Set operators	IN	\$intersect\$, \$union\$	intersect, union, except, exists, in, for all	|, :, &, -
Wildcard characters	\$	*	%, #	ˆ
Regular expression operators	*(Kleene-star), |(Alternation), +(One or more occurrences), .(Concatenation)	N/A	*(0 or more occurrences), +(1 or more occurrences), ?(0 or 1 occurrences), |(Disjunction)	*(Kleene-star), +(One or more occurrences), ?(0 or 1 occurrences)
Path Expression operators	N/A	/, //	.	(Implicit)
Quantifiers	N/A	\$any\$, \$all\$	some, any, all	N/A
Aggregate operators/methods	N/A	count()	min, max, count, sum, avg	N/A
Joins between DTDs (some/all extent)	Supported, see Section (5.1)	Not supported	Supported	Supported, see Section (4.3.3)

Comparisons of XML-QL, XQL, Lorel and the query language basis of QT4XML

5.5 Other XML Query Languages

XML-GL [Ceri et al., 1998] is a graph-based query language with both its syntax and semantics defined in terms of graph structures and operations. It allows the formulation of queries to extract information from XML documents and for restructuring such information into novel XML documents.

XQuery [DeRose, 1998] is not based on any traditional query language, rather its syntax is built directly on XPointer [DeRose, 1999] as XPointer is specifically designed for XML and has proven reliable and robust over a long period amongst other reasons to be chosen as a basis for XQuery.

5.6 Query Languages for Semistructured data

There have been several query languages for a semistructured data model and for variations of that model, a few of which are UnQL [Buneman et al., 1996], StruQL [Fernandez et al., 1997], Lorel [Abiteboul et al., 1996].

StruQL is query language for a web-site management system, which also has semistructured data, whereas, Lorel and UnQL focus on semistructured data in general. What is common across these languages is the fact that they need to query data at arbitrary depth. Superficially, we can think of a semistructured data model as an Object Oriented data model and use the features of OQL. However we then have some problems of querying missing data, querying any combination of values (heterogeneous data), etc. Also restructuring of data or information after querying is an issue of concern. StruQL allows construction of an arbitrary new graph, which is vital for creation of Web sites (the basic reason why StruQL is used). WebOQL [Arocena and Mendelzon, 1998] provides powerful features for tree and Web restructuring.

After looking at these query languages for semistructured data, there seems to be a need to define a general format for restructuring information, standardizing pointer traversals, querying to any depth, ordering of results, amongst other issues. These issues are relevant to querying XML data because XML represents semistructured data. Some of the query languages for XML have dealt with a few/all of the above issues as seen in the earlier section.

6 Conclusions and Future Work

XML is likely to play a central role in future information systems. It will be used for data interchange and XML documents will need to be stored in databases. This paper has presented a simple query tool for querying XML documents. With knowledge of XML, complex queries can be formulated without resorting to using traditional query languages like SQL or OQL.

With XML fast becoming a standard for semistructured documents on the Web, there is a need for Web query languages which can allow users to query XML documents, without having to learn a new query language. QT4XML presents one such opportunity. It is based on the DTD for any XML document. It works with the assumption that a Web user would be familiar enough with XML to understand the visual representation of QT4XML. The user is relieved from knowing any specific syntax for QT4XML as the query is built very intuitively. Also it allows constraints to be formed using regular expression operators and wildcard characters which can be used very effectively to query documents where the exact query criteria is not known. Most complex queries can be built easily using QT4XML, as seen from the examples. The prototype implementation shows how runtime mapping between XML documents and Java objects can be done. This is useful from the point of view that most databases which can be used for storing semistructured data currently, can do so only in the form of objects.

The prototype needs further work which we summarize below.

- The query tool supports joins, but does not support querying using XLink and XPointer.
- The tool provides for querying using IDREF(s), but only as text strings. Whether to actually provide for traversal through IDREF(s) would depend on the database for which this is used.
- Currently all the attributes and element values are stored as strings in this prototype. There needs to be a way to identify from the DTD how an element/attribute value is best expressed and then to provide support for it. The Visual Query Builder however supports querying independent of the format of the actual storage of elements. It is when this query is translated to a specific language that the need for format specification arises. XML Schema-Structures [Beech et al., 1999] need to be used in place of DTDs which allow for more constraints on XML document data and its type and thus the ability to identify expected type for an element or attribute.
- Coercion can be provided in the translated query of the Visual Query Builder, such that any element can be compared with a value and the query is executed such that the value is located in either of the element's attributes or subelements. Something similar is provided in Lorel [Abiteboul et al., 1996].
- Support for ANY type of Element of a DTD. Currently this is not supported by the Visual

Query Builder.

- Provision of operators which allow for case insensitive querying [Robie et al., 1998].
- QT4XML will be utilized in the JSIM project [Miller et al., 1998]. JSIM is a Web-based simulation environment supporting the storage of simulation results as well as models in Web-accessible databases. With QT4XML, finding information about simulation models or results, can become a process of rapid query formulation, followed by browsing the query result, leading to query reformulation. From a simulationist's point of view, it would be similar to surfing the Web, except that the query tool would provide much better precision than today's search engines. Web-based simulation carried out at a variety of sites and even using different simulation products can exchange models and simulation results, if they agree to use XML. (Semantic issues in interoperability may be partially addressed through defining standard sets of DTDs like CML for Chemistry or BSML for Genetics.)

References

[Abiteboul et al., 1996] Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J. L. (1996). The Lorel Query Language for Semistructured Data. Technical report, Stanford University. http://www-db.stanford.edu/pub/papers/lorel96.ps.

[Aeronaut, 1996] Aeronaut (1996). Prod - Visual Query Builder. Technical report, Aeronaut. http://www.aeronaut.com.au/hys_html/hys/Prod0212.htm.

[Alschuler and Walter, 1998] Alschuler, L. and Walter, M. (1998). Surprise! Netscape Puts XML Support in Mozilla. Technical report, Netscape Corporation. http://www.xml.com/xml/pub/SeyboldReport/bul032601.html.

[Arocena and Mendelzon, 1998] Arocena, G. O. and Mendelzon, A. O. (1998). WebOQL: Restructuring Documents, Databases and Webs. In *Proc. Fourteenth International Conference on Data Engineering*, pages 22–33, Orlando, FL.

[Beech et al., 1999] Beech, D., Lawrence, S., Maloney, M., Mendelsohn, N., and Thompson, H. S. (1999). XML Schema Part 1: Structures, W3C Working Draft, 6-May-1999. Technical report, W3C. http://www.w3.org/TR/xmlschema-1/.

[Bertram, 1998] Bertram, C. (1998). Semantic Information Correlation of Distributed Heterogeneous Assets in InfoQuilt. Technical report, University of Georgia, CS-LSDIS. http://www.cs.uga.edu/~bertram/Publications/thesis.ps.gz.

[Bray et al., 1998] Bray, T., Paoli, J., and Sperberg-McQueen, C. M. (1998). Extensible Markup Language (XML) 1.0. Technical report, W3C. http://www.w3.org/TR/REC-xml, W3C Recommendation, 10-February-1998.

[Buneman, 1997] Buneman, P. (1997). Tutorial: Semistructured Data. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, pages 117–121, Tucson, AZ.

[Buneman et al., 1996] Buneman, P., Davidson, S. B., Hillebrand, G. G., and Suciu, D. (1996). A Query Language and Optimization Techniques for Unstructured Data. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 505–516, Montreal, Canada. http://www.research.att.com/ suciu/strudel/external/files/camerareadyfinal.ps.

[Cattell et al., 1997] Cattell, R., Barry, D. K., Bartels, D., Berler, M., Eastman, J., Gamerman, S., Jordan, D., Springer, A., Strickland, H., and Wade, D. (1997). *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann Publishers, Inc., San Fransisco, CA.

[Ceri et al., 1998] Ceri, S., Comai, S., Damiani, E., Fraternali, P., Paraboschi, S., and Tanca, L. (1998). XML-GL: a Graphical Language for Querying and Restructuring XML Documents. Technical report, University of Milano. http://www2.elet.polimi.it/people/ceri/Xml-gl.

[Deach, 1999] Deach, S. (1999). Extensible Stylesheet Language (XSL) Specification. Technical report, W3C. http://www.w3.org/TR/WD-xsl.

[DeRose, 1998] DeRose, S. J. (1998). XQuery: A Unified Syntax for Linking and Querying General XML Documents. Technical report, W3C. http://www.w3.org/TandS/QL/QL98/pp/xquery.html.

[DeRose, 1999] DeRose, S. J. (1999). XML XPointer Requirements, Version 1.0, W3C Note 24-Feb-1999. Technical report, W3C. http://www.w3.org/TR/NOTE-xptr-req.

[Deutsch et al., 1998] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., and Suciu, D. (1998). XML-QL: A Query Language for XML. Technical report, W3C. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819.

[EECS, 1999] EECS (1999). PgAccess: A Tcl/Tk PostgreSQL Interface. Technical report, University of California at Berkeley. http://flex.flex.ro/pgaccess.

[Ernst, 1997] Ernst, J. (1997). Introduction to CDIF. Technical report, Electronic Information Association (EIA). http://www.eia.org/eig/cdif/intro.html.

[Fernandez et al., 1997] Fernandez, M., Florescu, D., Kang, J., Levy, A. Y., and Suciu, D. (1997). STRUDEL: A Web-site Management System. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 549–552, Tucson, AZ.

[Fujitsu, 1999] Fujitsu (1999). Fujitsu's 'HyBrick' SGML/XML Browser. Technical report, Fujitsu. http://www.fsc.fujitsu.com/hybrick.

[Goldman et al., 1999] Goldman, R., McHugh, J., and Widom, J. (1999). From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. Technical report, Stanford University. http://www-db.stanford.edu/pub/papers/xml.ps.

[Harold, 1998] Harold, E. R. (1998). *XML: Extensible Markup Language*. IDG Books Worldwide, Inc., Foster City, CA.

[IBM, 1999] IBM (1999). XML Parser for Java. Technical report, IBM. http://www.alphaworks.ibm.com/tech/xml.

[Laforge, 1998] Laforge, B. (1998). Coins: Connecting XML Elements and JavaBeans. Technical report, JXML, Inc. http://www.jxml.com/coins/index.html.

[MetaModel.com, 1999] MetaModel.com (1999). Metamodel Information. Technical report, Metamodel. http://www.metamodel.com.

[Miller et al., 1998] Miller, J., Ge, Y., and Tao, J. (1998). Component-Based Simulation Environments: JSIM as a Case Study Using Java Beans. In *Proc. 1998 Winter Simulation Conference*, pages 786–793, Washington, DC.

[Murray-Rust, 1998] Murray-Rust, P. (1998). The Jumbo XML/CML Browser. Technical report, XML-CML. http://www.xml-cml.org/jumbo.

[ODI, 1998] ODI (1998). Server-Side XML: Taming the Tower of Babel. Technical report, Object Design. http://www.odi.com/ODILIVE/contents/happenings/tech_trends/whitepapers/ServerSideXML.pdf.

[ODI, 1999] ODI (1999). An XML Data Server For Building Enterprise Web Applications. Technical report, Object Design. http://www.odi.com/ODILIVE/contents/happenings/tech_trends/whitepapers/XMLEnterpriseWebApps.pdf.

[ODMG, 1998] ODMG (1998). Object Data Management Group. Technical report, ODMG. http://www.odmg.org.

[Oracle, 1998] Oracle (1998). XML Support in Oracle8i and Beyond. Technical report, Oracle. http://www.oracle.com/xml/documents/xml_twp.

[Poet, 1997] Poet (1997). XML - The Foundation for the Future. Technical report, Poet. http://www.poet.com/products/cms_solutions/white_papers/foundation_for_the_future/index.html.

[Robie et al., 1998] Robie, J., Lapp, J., and Schach, D. (1998). XML Query Language (XQL). Technical report, W3C. http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[Thierry-Mieg and Durbin, 1992] Thierry-Mieg, J. and Durbin, R. (1992). Syntactic Definitions for the ACeDB Data Base Manager. Technical report, MRC Laboratory for Molecular Biology, Cambridge, England.

[Turner and Friedman, 1999] Turner, D. and Friedman, W. (1999). Getting Ready for Internet Explorer 5: XML and DHTML Enhancements. Technical report, Microsoft. http://msdn.microsoft.com/workshop/xml/articles/ICPXML.asp#_top.

[Ullman and Widom, 1997] Ullman, J. D. and Widom, J. (1997). *A First Course in Database Systems*. Prentice Hall, Upper Saddle River, NJ.

[Vatton et al., 1999] Vatton, I., Guétari, R., Kahan, J., and Quint, V. (1999). Amaya Overview. Technical report, W3C. http://www.w3.org/Amaya.

[Zloof, 1975] Zloof, M. (1975). Query By Example. In *Proc. NCC 44*, Anaheim, CA. AFIPS Press.

A Appendix: QT4XML Grammar

This appendix is used to provide the production rules for QT4XML. Some of the production rules are adapted from OQL [Cattell et al., 1997]

string ::= A non-empty string without any '-' | '*' | '?' | '\' | '|' | ':' | '^' | '+' | '&' | '"' or white space. If these characters or operators are to be included, they need to be escaped by preceding them with the escape character. To include white space, or any other string to be considered with its ordinary meaning, it needs to be enclosed within double quotes.

escape_char ::= ‘\’

wildcard_char ::= ‘^’

quote_char ::= ‘"’

closure_op ::= ‘*’ | ‘+’ | ‘?’

add_set_op ::= ‘|’ | ‘:’ | ‘-’

mult_set_op ::= ‘&’

pattern ::= string | ‘^’

expr ::= expr add_set_op term | term

term ::= term mult_set_op factor | term factor | factor

factor ::= factor closure_op | pattern | ‘(’ expr ‘)’

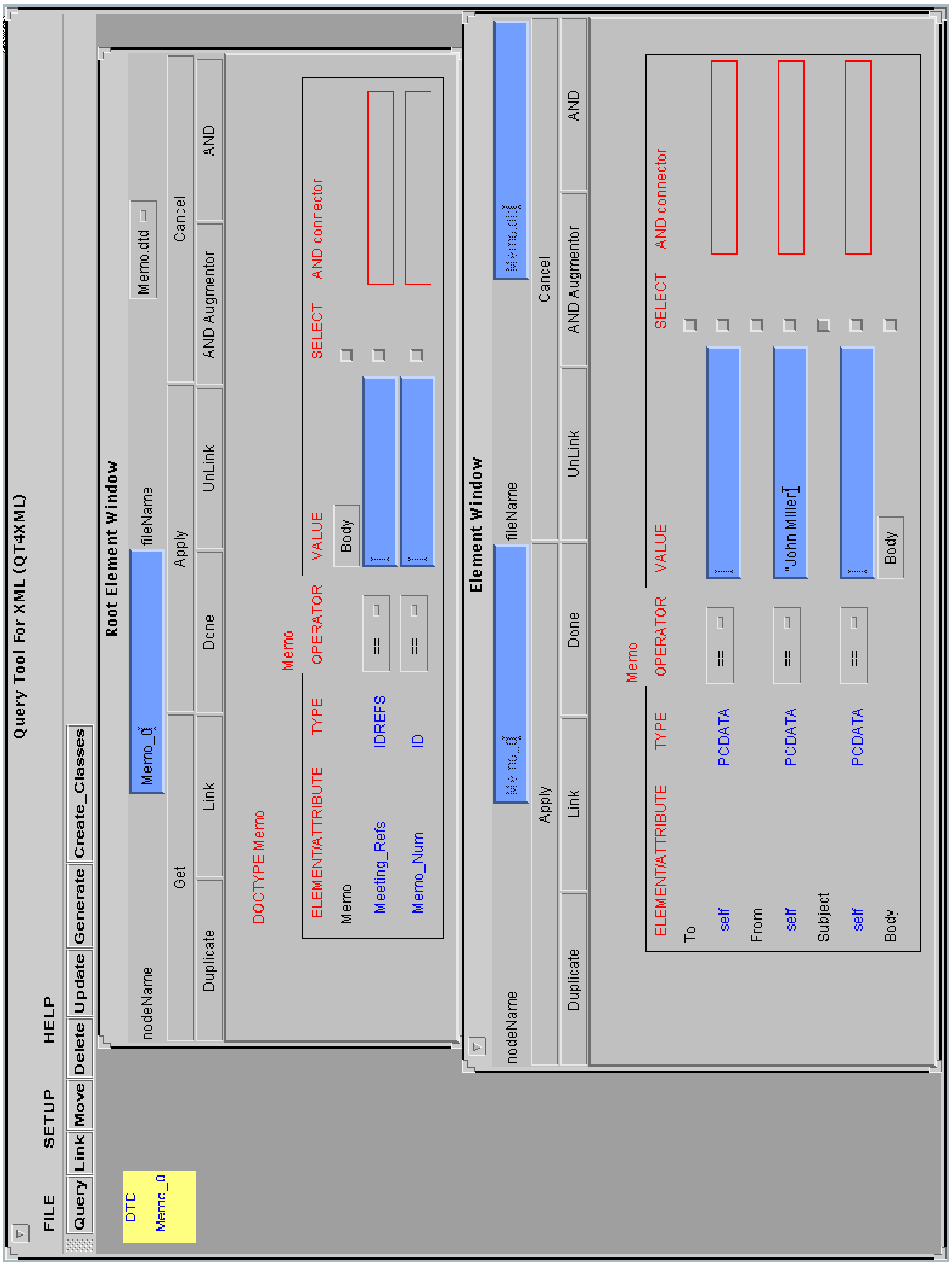

Figure 7: Query to Find Memos From John Miller

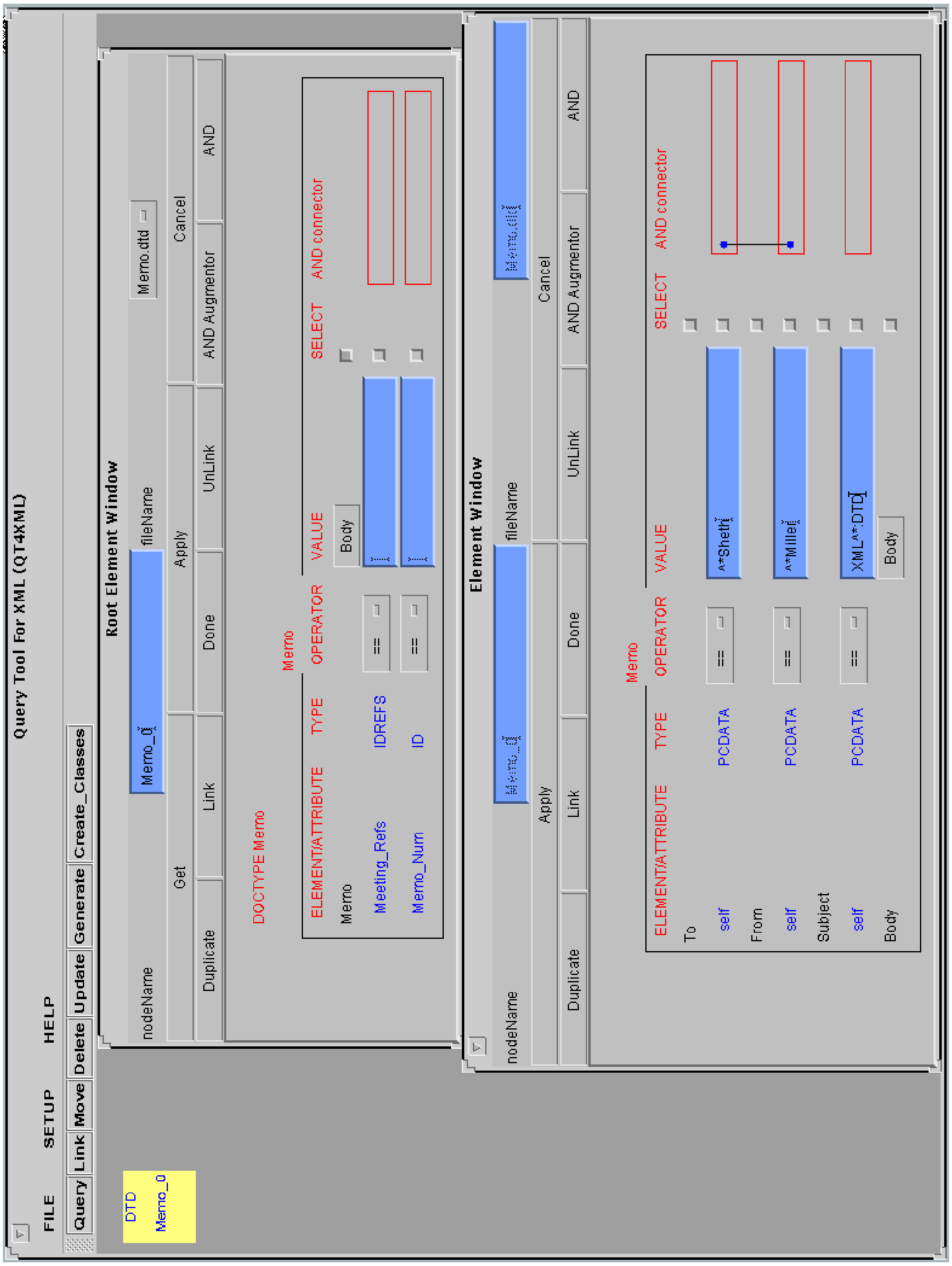

Figure 8: Query to Find Memos from Miller and to Sheth or with constraints on Subject

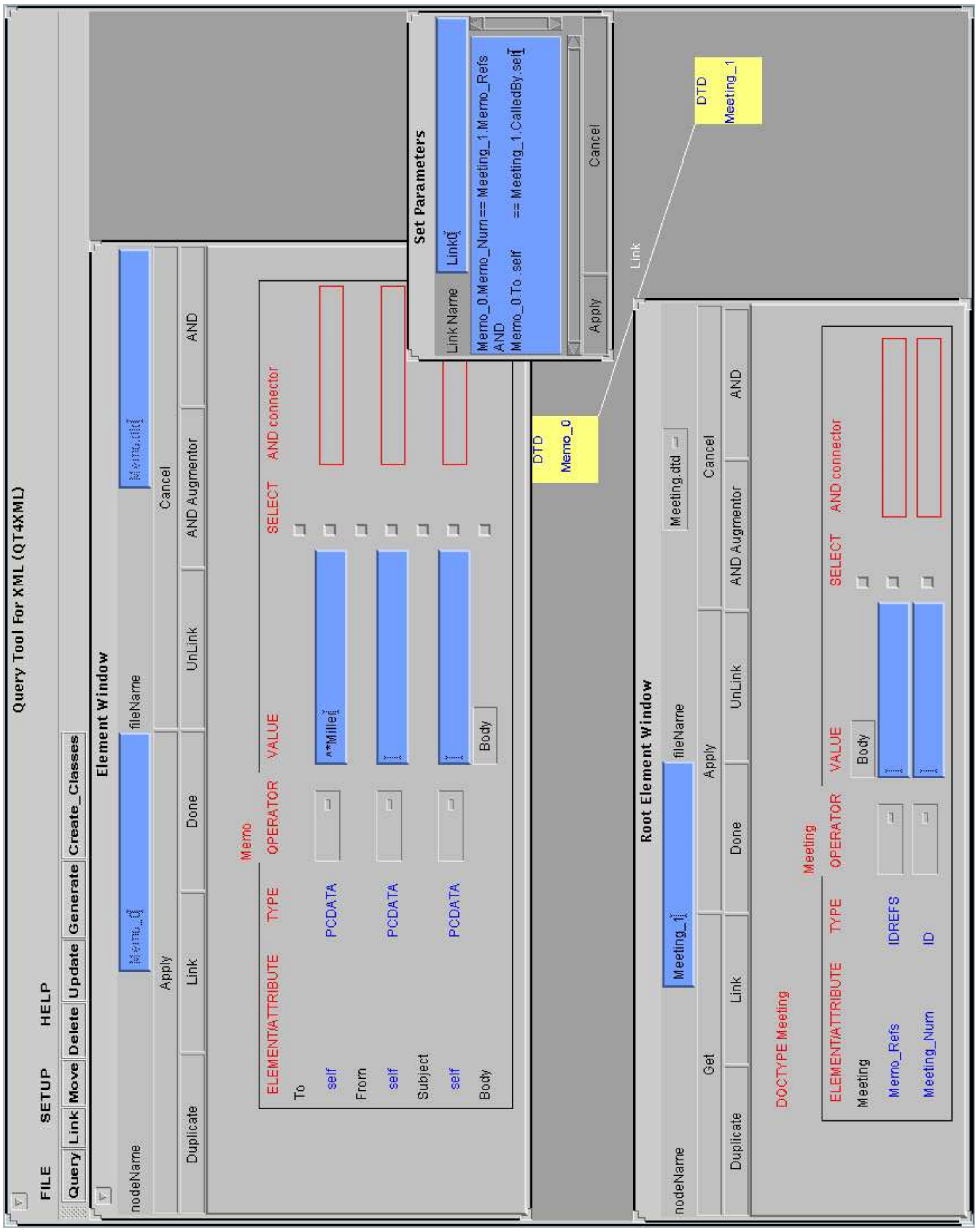

Figure 9: Detailed view of query: Find the memos where the person who received the memo is the person who called the meeting for discussing the memo and whose name ends with "Miller"