1 Query Languages

This section lists all the query languages that the Query Language Investigation team determined were relevant possibilities:

1.1 XPath

Reference: http://www.w3.org/TR/xpath20/

1.2 XQuery

Reference: http://www.w3.org/XML/Query

1.3 XQL

Reference: http://www.ibiblio.org/xql/xql-proposal.html

1.4 OQL

Reference: http://www.odmg.org/

1.5 SQL

Reference: http://www.ansi.org/

1.6 DMQL2

Reference: http://www.rets-wg.org/docs/index.html

2 XPath Query Language Comparison

2.1 XPath Language Definition

XPath was designed to be used (by XSLT and XPointer) to address parts of an XML document either by location or by pattern matching capabilities exercised on content at those locations. Pattern matching is facilitated via string manipulation, numeric operations, and Boolean logic functionality. The language uses a compact, non-XML syntax that operates on the hierarchical structure of an XML document. Each XPath expression operates starting from the current context (a node within the document) and yields a node set, a boolean value, a number, or a string; expressions can be nested.

The language consists of 37 tokens and 27 functions. 13 of the tokens are related to location, 14 to pattern matching, and 10 to the lexical structure. The functions deal with node identification, string manipulation, conversion to and numeric manipulation, and Boolean evaluation.

Given that it operates on an XML document, to be used as the RETS query language between a client and a server XPath would require that the client and server share a common XML document representation of the data to be searched. In essence that would have to be either a single document model for the entire site, one for each resource, or one for each class in each resource. Either separate document models or universally applied element attributers would have to exist for each set of names (System, Standard, etc). However, the Xpath language could be used to implement almost all of the functionality currently available including SearchType, Class, Count, Select, Limit, Offset, Standard Names and the Query itself with only Restricted falling outside the techniques inherent in the language.

Although all the DMQL Query functionality could be expressed in Xpath, some it it would be quite cumbersome since there are only two datatypes: string and numeric. All other datatypes would have to be supported by treating them as strings, use of the string parsing functions, and conversions to numeric for any query comparisons. The concept and implementation of lookup/expansion values would need to be significantly modified.

2.2 Available Parsers for XPath

A number of XPath parsers or rather Xpath parsing tool subsets are available in a variety of languages including Java, Python, LISP, C, C++, and inside .NET. Unfortunately, most of these tools only work on XML documents. Conversion to other query languages like SQL, especially for the syntax to handle non-numeric and non-string datatypes, would be a major effort.

Since it works inside a Document Object Model structure, performance is on par with interpretive LISP processors and degenerates exponentially with the size of the document.

2.3 Is XPath Human Readable?

Xpath intertwines the functionality of the various search parameters so the it is inherently more difficult to decipher. The location syntax will make sense to those who understand url syntax for hierarchical directory access and the pattern matching syntax for any but the most straightforward of queries would require familiarity with code that nests functions as parameters to other functions.

2.3.1 XPath Examples

Simple to select all the nodes in Site's Residential Property where Status=Active:

Site/Property/Residential/Listing[//Site/Property/Residential/Listing/MLSInformation[ListingStatus="Active"]]

Complex examples of XPath are very difficult to find. As best as I can tell, this will select the ListingID, UpdateDate, PhotoCount, and PhotoUpdateDate for listings with Status=Active or (OffMarketDate=2004-01-01+ and not Status=Active:

Site/Property/Residential/Listing/ListingID[(//Site/Property/Residential/Listing/MLSInformation[ListingStatus="Active") OR
(//Site/Property/Residential/Listing/MLSInformation[ListingStatus<>"Active" AND
(Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate],1,4)) >= 2004) AND
(Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate],6,2)) >=1) AND
Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate],9,2)) >= 1))] |
Site/Property/Residential/Listing/ModificiationTimestamp(//Site/Property/Residential/Listing/MLSInformation[ListingStatus="Active") OR
(//Site/Property/Residential/Listing/MLSInformation[ListingStatus<>"Active" AND
(Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate],1,4)) >= 2004) AND
(Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate],6,2)) >=1) AND
Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate],9,2)) >= 1))] |
Site/Property/Residential/Listing/PhotoCount(//Site/Property/Residential/Listing/MLSInformation[ListingStatus="Active") OR
(//Site/Property/Residential/Listing/MLSInformation[ListingStatus<>"Active" AND
(Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate],1,4)) >= 2004) AND
(Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate],6,2)) >=1) AND
Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate],9,2)) >= 1))] |

Site/Property/Residential/Listing/PhotoModificiationTimestamp(//Site/Property/Residenti
al/Listing/MLSInformation[ListingStatus=”Active”) OR
(//Site/Property/Residential/Listing/MLSInformation[ListingStatus<>”Active” AND
(Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate]
,1,4)) >= 2004) AND
(Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate]
,6,2)) >=1) AND
Number(Substring(//Site/Property/Residential/Listing/MLSInformation[OffMarketDate],
9,2)) >= 1))]

2.4 Is XPath a Superset of DMQL?

As indicated above, while virtually all of DMQL could be expressed, very complex expressions would be required to do something like a date range.

2.5 Does XPath Allow Joins?

Joins in this context (between two resources in separate XML documents) are not possible. Even a single document model does not support the traditional use of joins (to return a flat record) since it can only return already existing nodes. Either the document model would have to be “pre-joined” or the client would have to assemble the nodes into a flat document. In the latter case, a very convoluted syntax could retrieve nodes for the client from one section (Agents) based on content in another section (Agent-ID in Property).

2.6 XPath Ease of Implementation (Servers/Clients)

Unless the server is storing data in an XML document model and the client uses it in an XML document model, XPath would be very, very difficult to implement. Even inside an XML document model, since the functionality is a general search engine and not just specific data extraction, XPath poses many implementation challenges.

2.7 Ease of migration from DMQL2 to XPath

Migration from DMQL2 to XPath would not be easy—it is more likely that a developer would start over from scratch than try to retool.

2.8 Benefits of XPath over DMQL2

In an all XML document world, XPath has the cachet of being an XML-related standard.

3 XQuery Query Language Comparison

3.1 XQuery Language Definition

XQuery is designed to meet the requirements identified by the W3C XML Query Working Group as specified in the XQuery 1.0: an XML Query Language document. It is intended to be a language in which queries are concise and easily understood. It is also flexible enough to query a broad spectrum of XML information sources, including both XML documents and relational databases whose structure – nested, named trees with attributes – is similar to XML. XQuery is derived from an XML query language called Quilt, which in turn borrowed features from several other languages, including XPath 1.0, XQL, XML-QL, SQL and OQL.

The basic building block of XQuery is the expression, which is a string of Unicode characters. The language provides several types of expression that may be constructed from keywords, symbols, and operands. In general, the operands of an expression are another expression. Everything in XQuery is an expression that evaluates to a value. In addition, XQuery is a case-sensitive language.

XQuery has functions for numerous operations including date and time comparisons, mathematical calculations, string manipulations, and boolean algebra. If a particular function is needed that does not exist, it can be written. XQuery also consists of primitive data types, nodes and expressions, and sequences. The primitive data types in XQuery are:

- Numbers, including integers and floating-point numbers.
- The boolean values true and false.
- Strings of characters. These are immutable – i.e., you cannot modify a character in a string.
- Various types to represent dates, times, and durations.
- A few XML-related types. For example a *QName* is a pair that consists of a local name (like `template`) and a URL, which is used to represent a tag name like `xls : template` after it has been namespace-resolved.

As far as RETS implementation is concerned, we arrive at the same issue as XPath. Since XQuery operates using XML data, in order to be used as the RETS query language between a client and a server XQuery would require that the client and server share a common XML document representation of the data to be searched. In essence, that would have to be either a single document model for the entire site, one for each resource, or one for each class in each resource. Either separate document models or universally applied element attributers would have to exist for each set of names (System, Standard, etc). However, the XQuery language could be used to implement almost all of the functionality currently available, including SearchType, Class, Count, Select, Limit, Offset, Standard Names and Query.

3.2 Available Parsers for XQuery

A few parsers exist for XQuery, but it appears as though they are available only in Java.

3.3 Is XQuery Human Readable?

XQuery is much like any other programming language. It incorporates data types, built-in and user-defined functions and expressions that are not hard to understand. It maintains a syntax all its own, but remains human readable.

3.3.1 XQuery Examples

Situation:

List the MLNumber and MarketingRemarks of all listings that have a Status of “S” and contain “Ocean View” in the MarketingRemarks, ordered by MLNumber.

Solution in XQuery:

```
<result>
  {
    for $i in doc("listings.xml")//item_tuple
    where $i/status = “S”
      and contains($i/marketingremarks, "ocean view")
    order by $i/mlnumber
    return
       <item_tuple>
          { $i/mlnumber }
          { $i/ marketingremarks }
       </item_tuple>
  }
</result>
```

Expected Result:

```
<result>
   <item_tuple>
      <mlnumber>1003</mlnumber>
      <marketingremarks>Home has a great ocean view, complete with outdoor
fireplace.</marketingremarks>
   </item_tuple>
   <item_tuple>
      <mlnumber>1007</mlnumber>
      <marketingremarks>Enjoy an ocean view under beautiful oak
trees.</marketingremarks>
   </item_tuple>
</result>
```

More examples can be seen in the XML Query Use Case document prepared by W3C.

3.4 Is XQuery a Superset of DMQL?

Virtually all of DMQL could be expressed using XQuery. XQuery would also allow for arithmetic and boolean logic.

3.5 Does XQuery Allows Joins?

Joins are possible in XQuery. Here is an example from the XML Query Use Case document prepared by W3C, example 1.4.4.5 Q5:

Situation:

For bicycle(s) offered by Tom Jones that have received a bid, list the item number, description, highest bid, and name of the highest bidder, ordered by item number.

Solution in XQuery:

```
<result>
 {
  for $seller in doc("users.xml")//user_tuple,
      $buyer in  doc("users.xml")//user_tuple,
      $item in  doc("items.xml")//item_tuple,
      $highbid in  doc("bids.xml")//bid_tuple
  where $seller/name = "Tom Jones"
    and $seller/userid  = $item/offered_by
    and contains($item/description , "Bicycle")
    and $item/itemno  = $highbid/itemno
    and $highbid/userid  = $buyer/userid
    and $highbid/bid = max(
                        doc("bids.xml")//bid_tuple
                          [itemno = $item/itemno]/bid
                    )
  order by ($item/itemno)
  return
     <jones_bike>
        { $item/itemno }
        { $item/description }
        <high_bid>{ $highbid/bid }</high_bid>
        <high_bidder>{ $buyer/name }</high_bidder>
     </jones_bike>
 }
</result>
```

The above query does several joins, and requires the results in a particular order. If there were no order by clause, results would be reported in document order. If you do not care about the order, you can use the unordered function to inform the query processor that the

order of the lists in the for clause is not significant, which means that the tuples can be generated in any order. This can enable better optimization.

Expected Result:

```
<result>
   <jones_bike>
      <itemno>1001</itemno>
      <description>Red Bicycle</description>
      <high_bid>
         <bid>55</bid>
      </high_bid>
      <high_bidder>
         <name>Mary Doe</name>
      </high_bidder>
   </jones_bike>
</result>
```

3.6 XQuery Ease of Implementation (Servers/Clients)

Unless the server is storing data in an XML document model and the client uses it in an XML document model, XQuery would be very difficult to implement. The XQuery 1.0 document itself is 180 pages, and the Use Cases document is 81 pages. Both documents are quite thorough in their content, but there is a lot of material to cover.

3.7 Ease of Migration from DMQL2 to XQuery

Migration from DMQL2 to XQuery would not be easy. A developer would have to start over from scratch rather than try to retool.

3.8 Benefits of XQuery over DMQL2

XQuery hands down has more functionality than DMQL2. But the point to be made is that we are not maintaining XML documents of our databases. If it were a XML document world, XQuery would be a good option as a RETS query language.

4 XQL Alternative Query Language Investigation

4.1 XQL Language Definition

XQL originated in the late 1990s as one of the many attempts to define a query language that is suitable for searching databases that are organized using a XML structure. It is intended to meet the needs of application that have simple search requirements in a XML only environment. I don't believe that RETS Version 2 matches that set of requirements.

4.2 Available Parsers for XQL

There are a number of parsers available for Java, C, C++ and the .net environment and others. However there is not significant new activity as this language appears to have been bypassed for other alternatives. Given the limited amount of ongoing development, performance is not

4.3 Is SQL Human Readable?

This language is relatively ready by a technologist but a untrained person would not find it easily understandable.

4.3.1 XQL Examples

No examples are provided because of the unsuitable of the language. The Xpath examples illustrate this point.

4.4 Is XQL a Superset of DMQL?

No, the defined XML environment does not exist in the RETS structure and version 2 is not likely to move in the required direction.

4.5 Does XQL Allows Joins?

Joins of resources defined in different XML document, in a single operation, do not appear possible, using XQL. The language was not designed to meet that need and no apparent effort has been made to create extensions (which would be quite difficult).

4.6 XQL Ease of Implementation (Server/Client)

The requirement that the server have all of its accessible data in XML defined structures makes implementation unlikely in foreseeable future. In the real world that is not likely to ever occur. This is further complicated by the level of compliance required to any standard capable of obtaining even a minimal level of cross platform capabilities.

4.7 Ease of Migration to XQL from DMQL2

Migration is not practical, but would require a new implementation.

4.8 Benefits of XQL over DMQL2

In a all XML defined world there may be some benefits, but if that happened other query languages will better match the needs . In the RETS Version 2 environment there is no future for XQL as the primary search language..

5 OQL Query Language Comparison

5.1 OQL Language Definition

This section contains a brief description of the language.

The acronym OQL stands for Object Query Language which is a subset of the Object Data Management Group (ODMG) 2.0 or 3.0 specification.

This is a declarative (nonprocedural) language for querying and updating objects. The SQL-92 standard was used as the basis for OQL. It also includes support for object sets and structures. It also has object extensions to support object identity, complex objects, path expressions, operation invocation, and inheritance.

The details of this language specification can be found at: http://www.odmg.org. An online grammar definition can be found at: http://www.castor.org/oql.html, this is a subset of the full OQL language that does not implement features that can't be implemented in a single SQL statement.

5.2 Available Parsers for OQL

Castor provides an OQL to SQL translator (http://www.castor.org/oql.html). This parser actually creates a ParseTree which can likely be used to turn OQL into anything desired by the user. It can also take the OQL directly to SQL assuming that objects map directly to SQL tables.

The parser that is available removes the following symbols from the OQL Syntax:

andthen - Cannot be implemented in a single SQL query.
orelse - Same as above.
import - This is advanced functionality which may be added later.
Defined Queries – This is advanced functionality which may be added later.
iteratorDef – This was simplified to eliminate expressions.
objectConstruction – Removed due to Java issues.
structConstruction – Removed due to Java issues.

It also adds the following symbols:

between – similar to the SQL between operator.

This is the only Parser that I was able to find on the internet. It also seems that this is not fully implemented as there is still a Phase 4 of development to take place.

5.3 IS OQL Human Readable?

Since OQL is based on SQL, it is quite readable. I did find, however, that it was easy to get this much more complicated than SQL which, although readable, limited it's understandability.

5.3.1 OQL Examples

In order to quickly see what this language would look like, take the following OQL example:

```
select p.address from Person p
```

To help us understand what this really means, it is helpful to look at the SQL that the OQL to SQL translator would generate:

```
select address.* from person, address
where person.address_id = address.address_id
```

As one can see, it is a fairly economical language and allows expressions to be stated quite simply that actually result in some fairly complicated SQL.

Other examples that get slightly more complicated:

List the name and address of Guests with reservations for more than one day:

```
select struct(x.GuestName, x.StreetNr, x.City)
from x in Guest, y in x.has_Reservation
where y.NrDays > 1
```

Is there a reservation for the Kennedy room on 13 May?:

```
exists x in Reservation : x.ArrivalDate = "13 May"
and x.is_for_Room.RoomName = "Kennedy"
```

For each room, list the cities and arrival dates of guests with reservations:

```
select struct(x.RoomName,
(select struct(y.ArrivalDate, y.is_for_Guest.City
from y in x.is_reserved_in_Reservation))
from x in Room
```

Give the names of the Tutors which have a salary greater than $300 and have a student paying more than $30:

```
select t.name
from (select t from Tutors t where t.salary > 300 ) r,
     r.students s
where s.fee > 30
```

As can be seen from the above, while the basics are quite similar to SQL and the grammar is quite readable, the complexity can grow quite quickly. The language is geared towards a much move Object like representation of a query.

5.4 Is OQL a Superset of DMQL?

This is definitely a fact. All the basic features of DMQL and DMQL2 can be easily handled by this language definition. This is very evident after examining the language grammar provided by Castor.

5.5 Does OQL Allows Joins?

Being based on SQL, joins are fully supported in OQL.

5.6 OQL Ease of Implementation (Servers/Clients)

Server Side:

Since there is at least one parser available, it would not be too difficult to parse the language and build a server that could understand it and syntax check it.

However, there are so many features of this language, that I believe it would be an extremely complex task to build a server that could truly implement all the features and queries that this language allows. Things like object sets, structures, complex objects, path expressions, operation invocation, and inheritance would be an extremely complex set of features to implement fully.

Client Side:

Again, since there is at least one parser available, it would not be too difficult to build a client that could formulate a valid OQL query.

It would, however, be quite a complex task to build a user interface that would allow users to formulate queries that could take advantage of the power of the language.

5.7 Ease of migration to OQL from DMQL2

Based on the **OQL Ease of Implementation** section, I would say that the migration from DMQL would be extremely difficult if we were to try to implement the full functionality of the language.

If we, on the other hand, simplify it down to simply being able to specify a query on a single object or simply join of objects, the implementation would be fairly straightforward and would simply become a different way of expressing the syntax. In this case, it would be a very simple migration.

5.8 Benefits of OQL over DMQL2

There are a multitude of benefits. This is a very rich language that lets you do all that SQL can do and many more operations such as: some, any, all, for all. It also allows

collectors to be defined within statements such as array, set, bag and list. Several collection expressions are also available such as first, last, unique and exists.

This language definitely provides substantial benefits in terms of all the features that are available and the total control that it could give a client to specify exactly the query that is desired.

All of these benefits are quite tangible but, as can be seen by the **OQL Ease of Implementation** section, these all come at a significant implementation price on the server.

The premise for a language like OQL seems to stem from the need to allow programmers to stick to a similar syntax to overcome learning curve due to the impedance mismatch from relational database languages like SQL.

In our case, we are simply trying to define a language to allow a set of fairly simple objects to be queried in a fairly flexible way. I am not certain that we would gain anything over trying to make our queries more ‘Object like’.

Additionally, in order to be truly useful in our application, I believe that we would have to scale down the definition of the language into a very small subset of what it is truly capable of. Once this is done, it could be a nice way to implement a more object like query language but it would really then simply be a more interesting syntax than DMQL rather than adding any true benefits.

For these reasons, I would not recommend moving to OQL over DMQL.

6 SQL Query Language Comparison

6.1 SQL Language Definition

SQL is a standard, high-level query language that is used as the interface to relational databases. SQL can select, insert, update, and delete data, and there are commands to add or modify security, and to add and select from views (virtual tables). SQL has many functions,

SQL started out at IBM Research until 1986 where ANSI and ISO standardized the query language and was called SQL1 (AKA SQL-86). In 1992 SQL was revised and expanded to which was called SQL2 (AKA SQL-92). The current release, in 1999, ANSI and ISO released SQL99, also known as SQL3. Please visit http://www.ansi.org/ to purchase a copy of the standard.

6.2 Available Parsers for SQL

There are not a huge number of SQL parses. Mostly, I believe, because there are very few needs for such a thing. Most people parse from another query language (like DMQL2) into SQL (which is what their backend database uses). Searches on the Internet brought up a few sites including:

- Generic parser - http://www.antlr.org/ that has grammar for SQL here: http://www.antlr.org/grammar/1057863397080/index.html
- Java parser - http://www.experlog.com/gibello/zql/
- Delphi parser - http://sourceforge.net/projects/gasqlparser/
- Perl parser - http://search.cpan.org/~jzucker/SQL-Statement-1.005/lib/SQL/Parser.pm

6.3 Is SQL Human Readable?

SQL is designed to be very human readable. Using tokens like "SELECT", "FROM", "WHERE", "IN" to make most queries flow in a very human readable format. But, it is still possible to write a query that is very difficult to decode, with queries embedded in queries and complex joins.

6.3.1 SQL Examples

Select all field from property class 'RES' where the ListingStatus is 'Active'":
select * from PROPERTY where [Class]='RES' and [ListingStatus]='Active'

Select ListingID, UpdateDate, PhotoCount, and PhotoUpdateDate from property class RES where the status is either active or the status is not active and the OffMarketDate is greater then 01/01/2004:
select ListingID, UpdateDate, PhotoCount, PhotoUpdateDate from PROPERTY where [CLASS] = 'RES' and ([Status]='Active' or ([OffMarketDate] >= '2004-01-01' and [Status] != 'Active'))

Select all records from both property and agent where there is an agent ID in property that matches the AgentID in agent:
select * from [PROPERTY] join [AGENT] on [PROPERTY].[AgentID] = [AGENT].[AgentID]

6.4 Is SQL a Superset of DMQL?

Yes. All of DMQL and much more could be expressed using SQL.

6.5 Does SQL Allow Joins?

Joins are core piece of the SQL's query language

6.6 SQL Ease of Implementation (Servers/Clients)

Implementation on the Server side would be very complex. SQL allows embedded queries (queries inside queries) that can be processor intense, difficult to optimize and hard to interpret into the backend data storage system.

Implementation on the Client side is relatively easy, because of the flexibility the SQL gives.

6.7 Ease of migration to SQL from DMQL2

Migration from DMQL2 to SQL would probably require most people to retool. It would be possible for someone to write a DMQL2 to SQL converter (but it would be nearly impossible to go the other way around).

6.8 Benefits of SQL over DMQL2

The benefits are as follows:

- SQL is widely used - Most modern day relational database management systems use a SQL or SQL type query language
- SQL can be more human readable then DMQL2
- SQL has nearly all the required elements to do complicated multi-joins and pre-defined functions that any client would need.

7 DMQL2 Query Language – Suggested Extensions

7.1 DMQL2 Calculations

The ability to do field/field field/constant calculations in the DMQL2 query that would allow queries to contain simple calculations. Some examples are:

SalePrice + 10%

TODAY - 1

SalePrice .PLUS. [10%]

.TODAY. .MINUS. [1]

Note: .PLUS., .MINUS., .MUTIPLY., .DIVIDE. are new reserved words. Constants are enclosed in []

7.2 DLQL2 Less Than & Greater Than

Currently, the DMQL2 specification only allows the following kinds of queries:

The (-) sign indicates less than or equal to.
The (+) sign indicates greater than or equal to.

The (-) sign indicates less than or equal to.

The (+) sign indicates greater than or equal to.

< (less than)

> (greater than)

7.3 DMQL2 Aggregate Functions

It has been suggested that it would be handy to have the ability to perform aggregate functions such as SUM or AVERAGE for field values returned in a SELECT clause.

.SUM.(SalePrice) or maybe .SUM(SalePrice).

.AVERAGE.(SalePrice) or maybe .AVERAGE(SalePrice).

7.4 DMQL Cross Resource Joins

This section indicates whether joins between objects or table can be accomplished by the language. It should also indicate how intuitive the join syntax is.

In RETS the Select (Select=), from (Class=) and where (Query=) are handled as separate parameters. Both the SELECT and the QUERY need to be modified to handle a join. The easiest way to deal with this is to add a qualifier to the fieldname.

Example Join:

SearchType=Property
Class=1
Select=Agent.AgentName, Property.1.AgentID, Property.1.ListingID
Query=(Agent.AgentID = Property.1.AgentID), (AgentID='SMITH')

In the above example Property and Agent are the class name from the metadata (e.g. SearchType=). Since there is no qualifier for the last AgentID in the Query it is assumed to be from the specified (SearchType=)(Class=) (e.g. Property.1). This keeps everything backward compatible.