Use of XML based RETS Query Language in RETS 2.0

1 Query languages in RETS 1.X

RETS 1.X uses DMQL and DMQL2 as the query languages. While the DMQL(2) is compact and easy for developers to prepare, it has a number of deficiencies. First, because the query is not pre-parsed, a lot of effort has to be spent on the server to interpret the query. The parsing code can be fragile, and susceptible to reserved words, special characters, etc. Second, operators and operands are mixed and overloaded in DMQL(2), which makes it necessary to restrict characters allowed for field names and field values, or it will generate ambiguities. For example, the characters ",", "|", "+", "(" , ")" are not allowed to be contained in field names or field values. Third, there’s no standard way of validating queries.

Because of the similarities of DMQL and DMQL2, they’re referred to as DMQL hereafter in this whitepaper.

2 XML based RETS Query Language in RETS 2.0

2.1 The Language

As RETS 2.0 is moving towards fully XML and Web Services based, XML-based query language is needed to provide the consistency of the incoming and outgoing data format.

As an example, the DMQL query

\[(\text{Price}=100000-200000) , (\text{Area}=101,102) , (\text{ListingDate}=2004-10-10+) | (\text{Status}=Active))\]

can be expressed as:

```xml
<Query>
  <And>
    <Field name="Price" op="between">
      <Val>100000</Val>
      <Val>200000</Val>
    </Field>
    <Field name="Area" op="or">
      <Val>101</Val>
      <Val>102</Val>
    </Field>
    <Or>
      <Field name="ListingDate" op="greater">
        <Val>2004-10-10</Val>
      </Field>
      <Field name="Status" op="equal">
        <Val>Active</Val>
      </Field>
    </Or>
  </And>
</Query>
```

The XML RETS query language features a clear separation of operators and operands. The logical relationship among sub-queries is expressed using <And> and <Or> elements. Overall, this query language is designed for easy processing by RETS server software.
2.2 Benefits

- The query is pre-parsed, so the server can process the query without any ambiguity.
- The query is XML based, which makes it very easy for the server to validate using XML schema or other meanings of XML validation.
- Easy to add new features to the query language. For example, we can add the "select" parameter inside of the XML query if we choose to combine the search criteria and select statement.

2.3 Validation of the Query Language

Here's a generic XML schema that can validate the XML based query:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Query" type="QueryType"></xs:element>
  <xs:complexType name="ValueList">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element name="Val">
        <xs:simpleType>
          <xs:union memberTypes="xs:decimal xs:dateTime xs:string"></xs:union>
          <xs:attribute name="op"></xs:attribute>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="op"></xs:attribute>
    <xs:attribute name="name"></xs:attribute>
  </xs:complexType>
  <xs:complexType name="FieldList">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="Field" type="ValueList"></xs:element>
      <xs:element name="And" type="FieldList"></xs:element>
      <xs:element name="Or" type="FieldList"></xs:element>
    </xs:choice>
    <xs:attribute name="name" fixed="Price"></xs:attribute>
  </xs:complexType>
  <xs:complexType name="QueryType">
    <xs:choice>
      <xs:element name="And" type="FieldList"></xs:element>
      <xs:element name="Or" type="FieldList"></xs:element>
    </xs:choice>
  </xs:complexType>
</xs:schema>
```

More specific schema can be created based on server capability and metadata of a specific site. For example, the schema can express that a servers does not support nested queries. In that regard, the search validation schema will serve as a new metadata type that would allow client applications to verify a search locally before pass it over to the server.

2.4 Concerns

2.4.1 Data Size

The size of XML based query language is larger than DMQL. However with current hardware and software, the difference of memory expense and the process and transmission expense of the XML vs. DMQL is likely to be immeasurable. In most cases, the returned search data would be
many times bigger than the query language itself. So the overall impact of using XML as a query language will be negligible.

2.4.2 Readability
Comparing with DMQL, the XML query language may be a little harder to read for simple queries. But for more complex queries, particularly if nested queries are involved, DMQL is probably not going to be easier to understand than the XML query language.

In practice, end users will very rarely directly use the raw query language to access the server. Instead, they rely on client application’s user interface to help them to design the query. The generation of a pre-parsed query language should be as easy as the generation of DMQL for client applications. On the server, however, XML based query is easier to understand, and has no ambiguity.

Being in XML format, the query language can be easily converted to other query formats. As an example, an XSLT file that can covert the XML version to DMQL is appended in this article.

3 Conclusion

The addition of XML based query language will enable XML data format in round trip of RETS search transaction. It brings the benefits that XML has brought to other areas of the IT world. It is recommended an XML RETS Query language be developed and included in RETS 2.0 specification.

Appendix 1: XSLT file that converts XML based RETS query language to DMQL

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<stylesheet version="1.0" xmlns="http://www.w3.org/1999/XSL/Transform">
<output method="text" omit-xml-declaration = "yes"></output>
<template match="/Query">
  <apply-templates select="And|Or"></apply-templates>
</template>
<template match="And">
  <for-each select="And|Or|Field"><apply-templates select="."></apply-templates><if test="position() != last()">,</if></for-each>
</template>
<template match="Or">
  <text>(</text><for-each select="And|Or|Field"><apply-templates select="."></apply-templates><if test="position() != last()">|</if></for-each><text>)</text>
</template>
<template match="Field">
  <choose>
    <when test="@op='between'">
      <text>(</text><value-of select="@name"/>=<value-of select="Val[1]"/> and <value-of select="Val[2]"/>)</text>
    </when>
    <when test="@op='equal'">
      <text>(</text><value-of select="@name"/>=</value-of select="Val[1]"/>)</text>
    </when>
    <when test="@op='or'">
      <text>(</text><value-of select="@name"/>=</value-of select="Val[1]"/>)
    </when>
  </choose>
</template>
</stylesheet>
```
<text/>

(text在玩家的")><value-of select="@name"/>
<for-each select="Val" value-of select="."/> <if test="position() != last()">
</if></for-each></text>

<when test="@op='greater'">
<text/>
<when test="@op='greater'">
<text/>
<when test="@op='greater'">
<text/>
</when>
</choose>
</template>
</stylesheet>