Query Engines for Web-Accessible XML Data

Leonidas Fegaras  Ramez Elmasri
University of Texas at Arlington
Adding XML Support to an OODB

I will present:

- an extension to ODMG ODL, called \textit{XML-ODL};
- an extension to ODMG OQL, called \textit{XML-OQL};
- a mapping from XML-ODL to ODL;
- a translation scheme from XML-OQL queries into efficient OQL code.
Design Goals

We wanted to:

- provide full XML functionality to the data model and query language of an existing DBMS ($\lambda$-DB);
- provide uniform access of:
  - database data,
  - database-resident XML data (both schema-based & schema-less), and
  - web-accessible XML data (native form),
in the same query language;
- facilitate effective data storage and efficient query evaluation based on schema information (when available);
- provide clear, compositional semantics;
- avoid data translation.
Why Object-Oriented Databases?

- It is easier and more natural to map nested XML elements to nested collections than to flat tables;

- The translation of syntactic extensions into the core query language may create many levels of nested queries. But SQL supports very limited forms of query nesting, group-by, sorting, etc.

  e.g. it is difficult to translate an XML query that constructs XML elements on the fly into SQL.

- OQL can become a full-fledged XML query language with minimal effort. OQL already provides:
  - sorting,
  - arbitrary nesting of queries,
  - grouping & aggregation,
  - universal & existential quantification,
  - random access of list sub-elements.
Related Work

- Many XML query languages (XQL, Quilt, XML-QL, Lorel, Ozone, POQL, WebOQL, X-OQL,...) and a possible standard (XQuery).

- Some XML projects use OODB technology: Lore, YAT/Xyleme, eXcelon, ...

- More recently (June 2001), XQuery has been given typing rules and formal semantics (a mapping from XQuery to Core XQuery).
What is New Here?

- The design of our language extensions was driven by semantics, not standards.
- We provide complete, compositional semantics, which is also used as an effective translation scheme.
- In our semantics:
  - schema-less, schema-based, and web-accessible XML data, as well as OODB data, can be handled together in the same query;
  - schema-less queries do not have to change when a schema is given (static errors supersede run-time errors);
  - schema information, when provided, is utilized for effective storage and efficient query processing.
XML-OQL = OQL + Syntactic Extensions

Path expressions:

- e.A tag projection
- e._ any projection
- e.* wildcard projection (all descendants)
- e.@A XML attribute projection
- e[v→e'] filtering
- e[e'] indexing

Element construction:

- <tag> e₁, ..., eₙ </tag>

Entry points:

- retrieve(“handle”) database-resident XML data
- document(“file.xml”) native form
An XML-OQL Example

```xml
select <bib>
  <author> b.author.lastname </author>,
  <title> b.title </title>,
  <related>
    select <title> r.title </title>
    from r in b.@related_to
  </related>
</bib>
from b in document("bibliography.xml").bib.*.book
where b.year>1995
  and count(b.author)>2
  and b.title like "% Emacs %"
```

Result

```xml
<bib>
  <vendor id="id0_1"/>
  <name>Amazon</name>
  <email>webmaster@amazon.com</email>
  <book ISBN="0-8053-1755-4" related_to="0-7482-6284-4 07365-6522-7">
    <title>Learning GNU Emacs</title>
    <publisher>O'Reilly</publisher>
    <year>1996</year>
    <price>40.33</price>
    <author><firstname>Debra</firstname> <lastname>Cameron</lastname></author>
    <author><firstname>Bill</firstname> <lastname>Rosenblatt</lastname></author>
    <author><firstname>Eric</firstname> <lastname>Raymond</lastname></author>
  </book>
</bib>
```
Schema-Less (Generic) Mapping

A fixed ODL schema for storing schema-less XML data:

class XML_element ( extent Elements )
{
    attribute element_type element;
};

union element_type switch ( element_kind )
{
    case TAG: node_type tag;
    case PCDATA: string data;
};

struct node_type
{
    string name;
    list< attribute_binding > attributes;
    list< XML_element > content;
};
Translation of XML-OQL Paths

For example, e.A is translated into:

```sql
select y
from x in e,
    y in ( case x.element of
        PCDATA: list( ),
        TAG: if x.element.tag.name = "A"
            then x.element.tag.content
            else list() 
        end )
end )
```

Wildcard projection, e.*, requires a transitive closure (a recursive OQL function).
XML-ODL incorporates Xduce-style XML types into ODL:

() identity
A[t] tagged type
\{A_1:s_1,\ldots,A_n:s_n\} t type with attributes (s_1,\ldots,s_n are simple types)
t_1, t_2 concatenation
t_1 | t_2 alternation
t* repetition
t? optionality
any schema-less XML
integer
string

XML[t] may appear anywhere an ODL type is expected.
XML-ODL Example

```xml
<!ELEMENT bib (vendor*)>
<!ELEMENT vendor (name, email, book*)>
<!ATTLIST vendor id ID #REQUIRED>
<!ELEMENT book (title, publisher?, year?, price, author+)>
<!ATTLIST book ISBN ID #REQUIRED>
<!ATTLIST book related_to IDrefs>
<!ELEMENT author (firstname?, lastname)> 
```

```xml
bib[ vendor[ { id: ID } 
   ( name[string],
   email[string],
   book[ { ISBN: ID,
   related_to: bib.vendor.book.ISBN* } 
   ( title[string],
   publisher[string]?,
   year[integer],
   price[integer],
   author[ firstname[string]?,
   lastname[string] ]+ )
   ]* )
   ]* 
] 
```
XML-ODL to ODL Mapping

Some mapping rules:

\[
\begin{align*}
\text{[ A[t] ]} & \rightarrow [ t ] \\
\text{[ t₁, t₂ ]} & \rightarrow \text{struct } \{ [ t₁ ] \text{ fst; [ t₂ ] snd; } \} \\
\text{[ t₁ | t₂ ]} & \rightarrow \text{union (utag) } \{ \text{case LEFT: } [ t₁ ] \text{ left; case RIGHT: } [ t₂ ] \text{ right; } \} \\
\text{[ t* ]} & \rightarrow \text{list< [ t ] >}
\end{align*}
\]

If it has an ID attribute, \([ \{ A₁:s₁, ..., Aₙ:sₙ \} \ t ]\) is mapped to a class; otherwise, it is mapped to a struct.
**XML-OQL to OQL Mapping**

\[ [ t ]^x_{e.A} \] maps the XML path \( e.A \) into OQL code, given that the type of \( e \) is \( t \) and the mapping of \( e \) is \( x \).

Some mapping rules:

- \([ A[t] ]^x_{e.A} \rightarrow x\)
- \([ B[t] ]^x_{e.A} \rightarrow \text{empty}\)
- \([ t_1, t_2 ]^x_{e.A} \rightarrow \begin{cases} [ t_1 ]^{x.fst}_{e.A} \quad \text{if} \ [ t_2 ]^{x.snd}_{e.A} \text{is empty} \\ [ t_2 ]^{x.snd}_{e.A} \quad \text{if} \ [ t_1 ]^{x.fst}_{e.A} \text{is empty} \\ \text{struct} \{ \text{fst: } [ t_1 ]^{x.fst}_{e.A}; \text{snd: } [ t_2 ]^{x.snd}_{e.A}; \} \end{cases}\)
- \([ t^* ]^x_{e.A} \rightarrow \begin{cases} \text{empty} \quad \text{if} \ [ t ]^x_{e.A} \text{is empty} \\ \text{select} \ [ t ]^v_{e.A} \text{ from } v \text{ in } x \end{cases}\)

No searching (transitive closure) is needed for \( e.* \)
These are the XML documents stored in their native form at remote locations. For example,

`document("*").*.bib.*.author.*.name`

XML inverse indexes can be coded in ODL:

```plaintext
struct word_spec { doc, level, location };  
struct tag_spec { doc, level, ordinal, begin_loc, end_loc };  
class XML_word ( key word extent word_index )  
{ attribute string word;  
  attribute set< word_spec > occurs;  
};  
class XML_tag ( key tag extent tag_index )  
{ attribute string tag;  
  attribute set< tag_spec > occurs;  
};
```

XML-OQL path expressions over web-accessible XML data can now be translated into OQL code over these indexes.
Translating Web XML-OQL

The path expression $e.A$ is mapped to:

```
select y.doc, y.level, y.begin_loc, y.end_loc
from x in e,
    a in tag_index,
    y in a.occurs
where a.tag="A"
    and x.doc=y.doc
    and x.level+1=y.level
    and x.begin_loc<y.begin_loc
    and x.end_loc>y.end_loc
```

A typical query optimizer will use the primary index of tag_index (a B+-tree) to find the elements with tag “A”.

Much influenced by Niagara, but well integrated with the rest XML-OQL.
You May Ask …

- Compositional semantics is fine for proving the soundness of rules, but is it effective for implementing high-performance systems?
- Why don’t you use a semi-structured algebra?
- OODBs are a dead beat now. Why don’t you use a relational database?
- OK, this framework looks fine for XML-OQL. What about “real” XML query languages, such as XQuery?
- What about DTDs and XML Schemas?
Final Thoughts

- There are many benefits in storing and retrieving both XML and database data in the same system and language.

- There is common ground between XML Schema and ODL and between XQuery and OQL that we can take advantage of.

- OODB technology has a great potential for storing/retrieving XML data.

*Current status*: we have built a prototype system on top of λ-DB (an ODMG-based OODBMS).