Storage Techniques and Mapping Schemas for XML

Sihem Amer-Yahia
AT&T Labs – Research
sihem@research.att.com

Motivation and Outline

  - Flexibility of representation in XML induces multiple possible storage mappings into relational/object back-ends.
  - Mappings usually *hard-coded* into storage system and not easily accessible. Applications that need access to stored data would benefit from transparency.
- Capturing Identity, Structure and Order.
- Using Schemas (Schema-less, DTD, XML Schema).
- Commercial Tools.
- MXM.

Identity, Structure and Order

KFO: Foreign key in child element. ordinal value for sibling order (Edge, Attribute, Universal, LegoDB, commercial solutions). Key/Foreign key joins to recover document structure.

INTERVAL: Records subtree at each node. Includes intervals of descendants. Level number to distinguish children from descendants. Document ID to distinguish nodes from different documents. Stack-based joins with interval inclusion to recover structure (Timber).

DEWEY: Records at node path from the node to document root (LDAP). Stack-based joins with subtring comparisons to recover structure. Most complete but depends on depth in document.

Identity, Structure and Order: XRel

```
Path[pathID,pathexp]
Element[docID,pathID,start,end,index,reindex]
Attribute[docID,pathID,start,end,value]
Text[docID,pathID,start,end,value]
```

- Path stored as a string and substring matching used in joins.
- `start` and `end` record *region* of each node to distinguish between shared paths.
- `index` records document order and `reindex` records reverse document order.
- Reduces number of joins needed to recover document structure. Uses B+trees and Rtrees.

Using Schemas

Generic Mappings: No schema (Edge, Attribute, Universal). Element and Attribute relations.

Fixed Mappings: DTD-driven (Basic, Shared, Hybrid). One relation per element. Variations in inlining/outlining elements.

Flexible Mappings: Cost-based strategy (LegoDB). Uses *Inlining*, *Union Factorization*, *Repetition Merge*, *Wildcard rewritings* and *Unions to options* rewritings on XML Schema.

Commercial Tools

- Use a mapping language.
- Hard-code defaults such as KFO for document structure and attribute inlining.
- IBM supports storing XML documents in CLOBs with side tables to index structure.
- IBM and Oracle can validate stored documents. Microsoft?
- Tailored to one particular system. Cannot be used for other relational back-ends.

IBM DB2 XML Extender

- Annotate a simplified XML Schema with mapping information. DAD defines RDB_Nodes. Used for publishing and storage. Stored procedures to shred `dxxShredXML()`, and build documents `dxxGenXML()`. Other functions for type conversion, retrieval with Spath and `Update(xmlobj, path,value)`.
- XML DTD repository stores meta information on mappings. Users can access this table to insert their own DTDs. DTDs can be used to validate XML documents.

IBM DB2 XML Extender

```
<DAD>
 <Xcollection>
  <root_node>
    <element_node name=’’PATIENT’’>
      <attribute_node name=’’IDNum’’>
        <RDB_node>
          <table name=’’Patient_tab’’/>
          <column name=’’Patient_key’’/>
          <condition>IDNum > "635"</condition>
        </RDB_node>
      </attribute_node>
    </element_node>
  </root_name>
 </Xcollection>
</DAD>
```

Microsoft SQL Server

- 3 publishing modes: RAW, AUTO and EXPLICIT.
- 3 storage solutions: Edge, XSD and OpenXML.
- Annotated schemas XSD (successor to the XML-Data Reduced (XDR) schema definition language). Implemented in SQLXML which includes an XDR to XSD converter tool. Describes table and column names. Contains embedded SQL. Used both for publishing and for storage.
- OpenXML compiles XML documents into DOM. XPath used to decompose documents into tables.

```
Select * from OpenXML(@pat, ‘/HL7/PATIENT’, 1)
WITH (IDNum int GiNa varchar(20))
```

Microsoft SQL Server

```
<schema xmlns:xsd="http://www.w3.org/XMLSchema"
        xmlns:sql="urn:schemas-microsoft:mapping-schema">
 <element name="PATIENT" sql:relation="Patients">
  <complexType>
   <sequence name="PaNa">
    <element name="FaNa" sql:field="LastName" type="string" />
    <element name="GiNa" sql:field="FirstName" type="string" />
   </sequence>
   <attribute name="IDNum" sql:field="PatientID" type="integer"/>
  </complexType>
 </element>
 <annotation>
  <appinfo>
   <sql:relationship name="Patient_OBX" parent="Patients"
    parent-key="PatKey" child="OBX" child-key="OBXKey" />
  </appinfo>
 </annotation>
</schema>
```

Oracle 9iR2

- Oracle8i (1999) for XML publishing. Oracle9i (2001): XML Parsers, XSLT Processor, XML SQL to generate documents.
- New datatypes-for XML (XMLType) and for logical pointers (URI-Ref)-. Functions to shred documents into tables. Operators associated to XMLType: `Extract()` evaluates XPath expression, `existsNode()`.
- Oracle 9iR2 introduced Oracle XML DB. Mappings are system- or user-generated (by annotating an XML Schema).
- SQL to generate documents: `XMLELEMENT`, `XMLATTRIBUTES`. Lazy materialization reads DOM nodes on demand. XPath optimized using Btrees.

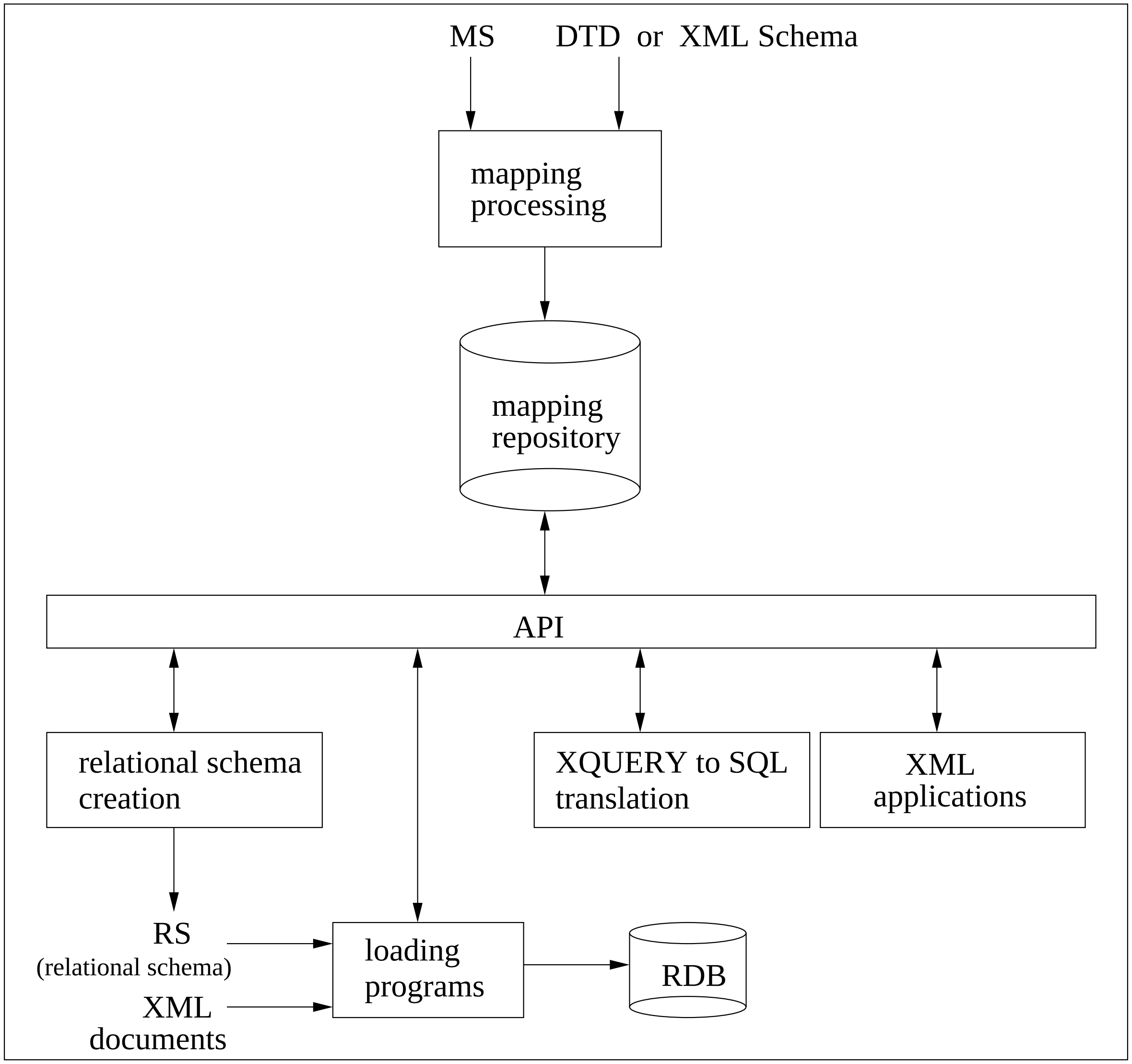
Oracle 9iR2

```
Declare doc varchar(5000) :=

‘<schema>
<complexType name = "PATIENT"
             xdb:SQLType = "OBJ_T2">
  <sequence>
    <element name = "FaNa" type = "string"/>
    <element name = "OBX" xdb:SQLType = "CLOB">
  </sequence>
</complexType>
</schema>’

Begin dbms_xmlschema.registerSchema(
‘http://www.oracle.com/HL7.xsd’, doc) end;
```

MXM: Goal and Design Choices

- Capture existing XML-to-Relational mappings and express them in a declarative language. Make them accessible through an interface. Do not hard-code defaults.
- Map schema-less documents, documents conforming to a DTD and documents conforming to an XML Schema.
- Orthogonal Design. Express existing mappings and more! XML Syntax. Extensible. Invertible?

MXM: Architecture



MS

DTD or XML Schema

mapping processing

mapping repository

API

relational schema creation

XQUERY to SQL translation

XML applications

RS (relational schema)

XML documents

loading programs

RDB

MXM: Example DTD

```
<!DOCTYPE addressBook [
<!ELEMENT addressBook addressBookContent>
<!ATTLIST addressBook owner CDATA>
<!ENTITY addressBookContent (fnameTelephone)*>
<!ENTITY fnameTelephone (fname,telephone)>
<!ELEMENT fullname (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>]>
```

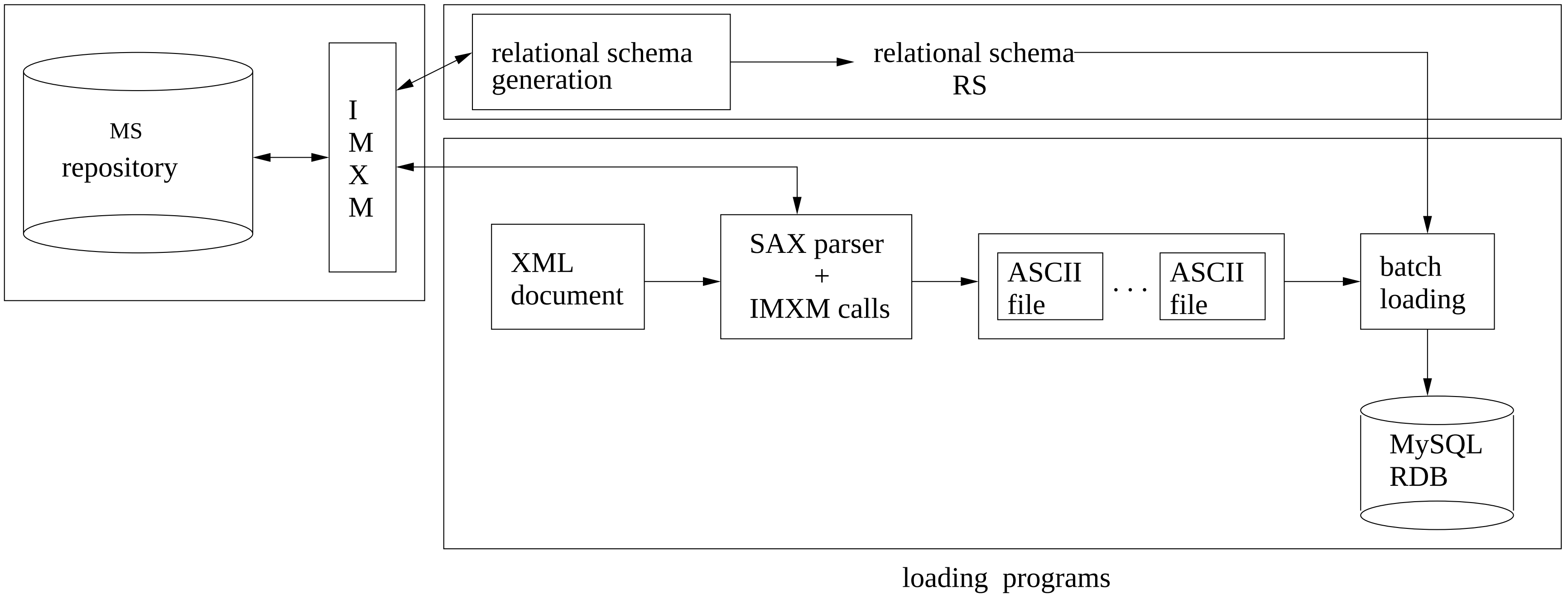
```
RaddressBook[addressBook_INTERVAL,tag,owner_VAL]
fullnameTable[fullname_INTERVAL,tag,fullname_VAL]
telephone[telephone_INTERVAL,telephone_VAL]
```

```
<XtoRMapping from=”XS” to=”RS1”>
    <StructMap whichMap=”INTERVAL”/>
    <TableMap>
        <table whichTable=”RaddressBook”>
            <sourceName>addressBook</sourceName>
        </table>
        <table whichTable=”fullnameTable”>
            <sourceName>fullname</sourceName>
        </table>
        <table whichTable=”telephone”>
            <sourceName>telephone</sourceName>
        </table>
    </TableMap>
    <CLOBMap/>
</XtoRMapping>
```

IMXM

- `getStructMap()`
  `isInlined(ElemName|AttName)`
  `getTableName(ElemName|AttName)`
  `getCLOBName(ElemName|AttName)`
  `getFields(TableName)`
  `getFieldType(FieldName)`
- Query defaults (e.g., `getDefTabNaming()`).
- Granularity of API depends on application (e.g., return all mapping information related to a particular element).

MXM: Implementation



MXM: Extensions and Applications

- Extend mapping specification to other backends (e.g., LDAP, native systems).
- Incorporate constraints on schemas (e.g., cardinality constraints in XML Schema).
- Implement mappings on top of commercial solutions.

Tuning mappings: Modify declarative mapping specification and test different XML-to-relational mappings.

XML data exchange: Avoid building entire XML documents and identify *fragments* of documents to exchange.