COMPSCI 732

NXDs - what are the other issues?

What other features of NXDs are important?

- Round tripping
- Role of schema
- Locking or access control
- Transactions
- Recovery
- Query optimization

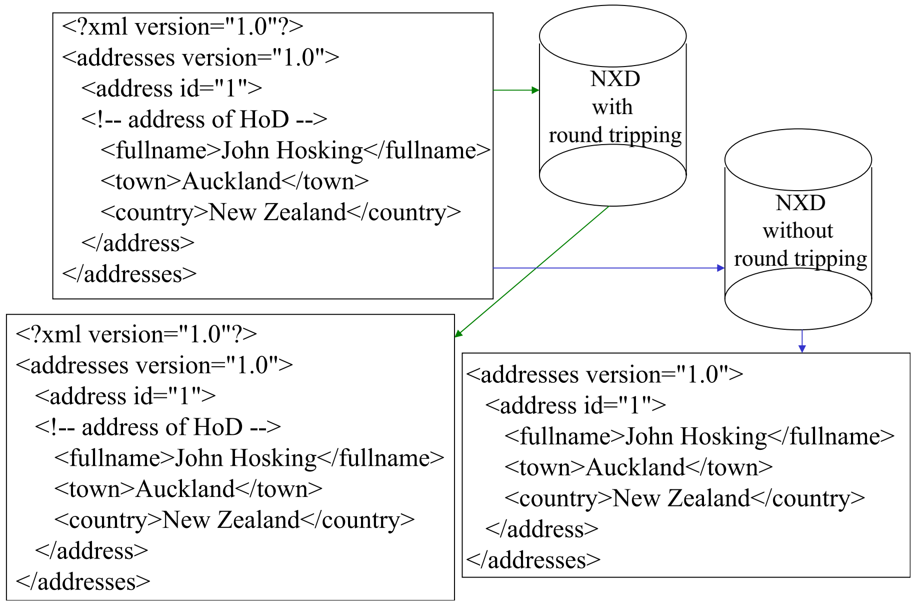
Round tripping

Definition: you can store an XML document in a native XML database and get the “same” document back again.

This is important:
•in document centric applications where CDATA sections, comments and processing instructions are an important part of the document.
•to many legal and medical applications, which are required by law to keep exact copies of documents.

All NXDs can round trip documents at the level of elements, attributes, PCDATA and document order.
As a general rule, text based NXDs round trip XML documents exactly,
While model based NXDs round trip XML documents at the level of their document model.

Example: Round tripping



```
<?xml version="1.0"?>
<addresses version="1.0">
  <address id="1">
  <!-- address of HoD -->
    <fullname>John Hosking</fullname>
    <town>Auckland</town>
    <country>New Zealand</country>
  </address>
</addresses>
```

```
<?xml version="1.0"?>
<addresses version="1.0">
  <address id="1">
  <!-- address of HoD -->
    <fullname>John Hosking</fullname>
    <town>Auckland</town>
    <country>New Zealand</country>
  </address>
</addresses>
```

```
<addresses version="1.0">
  <address id="1">
    <fullname>John Hosking</fullname>
    <town>Auckland</town>
    <country>New Zealand</country>
  </address>
</addresses>
```

Role of schemas in RDBs

- Checking consistency of database
- Physical storage e.g., fixed vs variable length records
- Query processing e.g., know structure of data and mapping to physical storage
- Query optimization e.g., derive integrity constraints such as primary key, foreign key etc.
- What else?

Schemas in NXDs

Some NXDs do not require a schema, e.g. Apache Xindice (pronounced zeen-dee-chay).

“You also gain a lot of flexibility through the semistructured nature of XML and the schema independent model used by Xindice.”

Other NXDs do require a schema, e.g., Tamino.

What do you gain and what do you lose?

Schema-less NXDs

•Gain
 •Flexibility
 •Don’t have to worry about schema evolution when XML document updated

•Lose
 •Consistency checking
 •Detail for query processing
 •Detail for mapping to physical storage

One possibility is to allow some kind of hybrid where user chooses where it is appropriate to specify schema and where it isn’t…

Schema languages that have been suggested

Schemas used to specify valid elements that can occur in a document, the order in which they occur and specify integrity constraints.
DTD – specify order and occurrence of elements BUT different syntax from XML, and doesn’t support data types.
XDR – same syntax as XML, and supports data types, submitted to W3C by Microsoft Corporation BUT rejected by W3C.
XML Schema – supports more data types than XDR, allows for the creation of custom data types.

DTD

SAMPLE XML FRAGMENT

```
<uoa_student num=“uoa000x">
   <name>George Burdell</name>
   <age>21</age>
</uoa_student>
```

DTD FOR SAMPLE XML FRAGMENT

```
<!ELEMENT uoa_student (name, age)>
 <!ATTLIST uoa_student num CDATA>
<!ELEMENT name (#PCDATA)>
<!ELEMENT age (#PCDATA)>
```

XDR (XML Data Reduced)

SAMPLE XML FRAGMENT

```
<uoa_student num=“uoa000x">
   <name>George Burdell</name>
   <age>21</age>
</uoa_student>
```

XDR FOR SAMPLE XML FRAGMENT

```
<Schema name="myschema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="age" dt:type="ui1" />
  <ElementType name="name" dt:type="string" />
  <AttributeType name="num" dt:type="string" />
  <ElementType name=“uoa_student" order="seq">
  <element type="name" minOccurs="1" maxOccurs="1"/>
  <element type="age" minOccurs="1" maxOccurs="1"/>
  <attribute type="num" />
  </ElementType>
</Schema>
```

XML Schema

XSD FOR SAMPLE XML FRAGMENT

```
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema" >
  <xsd:element name=“uoa_student">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type=“xsd:string"/>
        <xsd:element name="age" type=“xsd:unsignedInt"/>
      </xsd:sequence>
      <xsd:attribute name="num">
        <xsd:simpleType>
          <xsd:restriction base=“xsd:string">
            <xsd:pattern value=“uoa\d{3}[A-Za-z]{1}"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

What should happen when the schema changes?

- prevent users from changing DTDs/schema as long as documents exist in the collection (not flexible)
- refuse edit if any documents in collection fails to validate against the new schema (not flexible enough)
- delete parts of the instance documents that were referenced by the deleted parts of the schema (difficult to do!)
- Update the schema (how difficult?)

Summary

- XML was originally described as a meta-markup language. It soon became clear that XML also provided a way to describe data making it important as a data storage and interchange format.
- NXDs are not yet widely used, although there are open source and commercial implementations.
- Some NXDs are better for some applications:
  - How structured the data is
  - If data is data or document centric
  - Whether round tripping is important or not
  - Whether the main use is querying or updating the data
  - …
- There are still many open questions in the implementation and use of NXDs