XML Parsers
XPath, XQuery

Outline

- XML parsers
- XPath
- XQuery

- Background (reading)
  - http://www.w3.org/TR/xmlquery-use-cases/ several Xquery examples
  - http://www.galaxquery.org/ nice command line tool
XML Parsers - Overview

• What do we do if we need to “read” an XML document?
  – This is typically called parsing
    • Navigate through XML trees
    • Construct XML trees
    • Output XML
  – SOAP libraries use this technique to read and write XML messages

Two XML Parsers

• Two main APIs for XML parsers
  – **DOM** (Document Object Model)
    • Tree structure
  – **SAX** (Simple Api for Xml)
    • Read individual tags
• Built for programmers who do not want to write their own parsers
• Available for multiple languages
DOM

• Platform- and language-neutral interface
• Allows to build documents, navigate their structure, and add, modify, or delete elements and content.

DOM Example

```xml
<?xml version="1.0"?>
<addressbook>
  <person>
    <name>Nazmul Idris</name>
    <email>xml@java.xml.com</email>
  </person>
  <person>
    <name>John Doe</name>
    <email>john@doe.com</email>
  </person>
</addressbook>
```

Figure 1: Hierarchical structure of a document object
DOM Cont.

- **Tree** for XML works fine since XML is hierarchically organised
- Every XML document can be represented as a tree

**Some Insights in API**

```java
org.w3c.dom.Document

NodeList
getElementsByTagName(java.lang.String tagname)
   Returns a NodeList of all the Elements with a given tag name in the order in which they are encountered in a preorder traversal of the Document tree.

Element
getDocumentElement()
   This is a convenience attribute that allows direct access to the child node that is the root element of the document.
```

http://java.sun.com/webservices/jaxp/dist/1.1/docs/api/org/w3c/dom/package-summary.html
Some Insights in API

org.w3c.dom.Node

NodeList
getChildNodes()
A NodeList that contains all children of this node.

Node
getFirstChild()
The first child of this node.

Node
getLastChild()
The last child of this node.

Node getParentNode()
The parent of this node.
class ContractorLastNamePrinter {
    ContractorLastNamePrinter(Document doc) {
        System.out.println();
        try {
            //*** GET DOCUMENT ELEMENT BY NAME ***
            NodeList nodelist = doc.getElementsByTagName("workers");
            Element elm = (Element) nodelist.item(0);

            //*** GET ALL contractors BELOW workers ***
            NodeList contractors = elm.getElementsByTagName("contractor");
            for (int i = 0; i < contractors.getLength(); i++) {
                Element contractor = (Element) contractors.item(i);

                //*** NO NEED TO ITERATE info ELEMENTS, ***
                //*** WE KNOW THERE'S ONLY ONE ***
                Element info =
                    (Element) contractor.getElementsByTagName("info").item(0);
                System.out.println(
                    "Contractor last name is " + info.getAttribute("lname"));
            }
        } catch (Exception e) {
            System.out.println(
                "ContractorLastNamePrinter() error: " + e.getMessage());
        }
    }
}
SAX

• Access to XML information as a sequence of events
  – Document is scanned from start to end
• Faster than DOM
• You can create your own object model
• You are responsible to interpret all the objects read by the parser

SAX Events

• the start of the document is encountered
• the end of the document is encountered
• the start tag of an element is encountered
• the end tag of an element is encountered
• character data is encountered
• a processing instruction is encountered
<purchase-order>
  <date>2005-10-31</date>
  <number>12345</number>
  <purchased-by>
    <name>My name</name>
    <address>My address</address>
  </purchased-by>
  <order-items>
    <item>
      <code>687</code>
      <type>CD</type>
      <label>Some music</label>
    </item>
    <item>
      <code>129851</code>
      <type>DVD</type>
      <label>Some video</label>
    </item>
  </order-items>
</purchase-order>

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

private static final class SaxHandler extends DefaultHandler {
  // invoked when document-parsing is started.
  public void startDocument() throws SAXException {
    System.out.println("Document processing started");
  }
  // notifies about finish of parsing:
  public void endDocument() throws SAXException {
    System.out.println("Document processing finished");
  }
  // we enter to element 'qName':
  public void startElement(String uri, String localName, String qName, Attributes attrs) throws SAXException {
    if (qName.equals("purchase-order")) {
    } else if (qName.equals("date")) {
      /* if (...) */
      } else {
        throw new IllegalArgumentException("Element '" + qName + "' is not allowed here");
      }
  }
  // we leave element 'qName' without any actions:
  public void endElement(String uri, String localName, String qName) throws SAXException {
    // do nothing;
  }
}
Outline

• XML parsers
• XPath
• XQuery

Querying XML Data

• XPath = simple navigation through XML tree
• XQuery = the SQL of XML
• XSLT = recursive traversal
  – eXtensible Stylesheet Language Transformation
  – will not discuss

• XQuery and XSLT build on XPath
Sample Data for Queries

```
<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> Rick </author>
    <author> Hull </author>
    <author> Victor Vianu </author>
    <title> Foundations of Databases </title>
    <year> 1995 </year>
  </book>
  <book price="55">
    <publisher> Freeman </publisher>
    <author> Jeffrey D. Ullman </author>
    <title> Principles of Database and Knowledge Base Systems </title>
    <year> 1998 </year>
  </book>
</bib>
```

Data Model for XPath

```
<bib>
  <book>
    <publisher> </publisher>
    <author> </author>
    <author> </author>
    <author> </author>
    <title> </title>
    <year> </year>
  </book>
  <book>
    <publisher> </publisher>
    <author> </author>
    <author> </author>
    <author> </author>
    <title> </title>
    <year> </year>
  </book>
</bib>
```
XPath: Simple Expressions

/bib/book/year

Result:  

<year> 1995 </year>

<year> 1998 </year>

/bib/paper/year

Result: empty  (there were no papers)

XPath: Restricted Kleene Closure

//author

Result:  

<author> Serge Abiteboul </author>

<author> <first-name> Rick </first-name>

<last-name> Hull </last-name>

</author>

<author> Victoria Vianu </author>

<author> Jeffrey D. Ullman </author>

/bib//first-name

Result:  

<first-name> Rick </first-name>
XPath: Text Nodes

Result: Serge Abiteboul
       Jeffrey D. Ullman

Rick Hull doesn’t appear because he has firstname, lastname

Functions in XPath:
  – text()  = matches the text value
  – node()  = matches any node (= * or @* or text())
  – name()  = returns the name of the current tag

XPath: Wildcard

Result: <first-name> Rick </first-name>
        <last-name> Hull </last-name>

* Matches any element
XPath: Attribute Nodes

/bib/book/@price

Result: “55”

@price means that price is has to be an attribute

XPath: Predicates

/bib/book/author[first-name]

Result: <author> <first-name> Rick </first-name> <last-name> Hull </last-name> </author>

Predicate corresponds to an IF/THEN statement. If it is true, the Element will be selected!

General: parent[child someTestHere]
XPath: More Predicates

/bib/book/author[firstname][address[.//zip][city]]/lastname

Result: `<lastname> … </lastname>`

XPath: More Predicates

/bib/book[@price < “60”]

/bib/book[author/@age < “25”]

/bib/book[author/text()]
XPath: Summary

- `bib` matches a `bib` element
- `*` matches any element
- `/` matches the `root` element
- `/bib` matches a `bib` element under `root`
- `bib/paper` matches a `paper` in `bib`
- `bib//paper` matches a `paper` in `bib`, at any depth
- `//paper` matches a `paper` at any depth
- `paper//book` matches a `paper` or a `book`
- `@price` matches a `price` attribute
- `bib/book/@price` matches `price` attribute in `book`, in `bib`
- `bib/book[@price<"55"]/author/lastname` matches…

Outline

• XML parsers
• XPath
• XQuery
XQuery Motivation

• Query is a strongly typed query language
• Builds on XPath
• XPath expressivity insufficient
  – no join queries (as in SQL)
  – no changes to the XML structure possible
  – no quantifiers (as in SQL)
  – no aggregation and functions

FLWR (“Flower”) Expressions

• XQuery uses XPath to express more complex queries

```
for ...
let...
where...
return...
```
Sample Data for Queries

```xml
<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> Rick </author>
    <author> Hull </author>
    <author> Victor Vianu </author>
    <title> Foundations of Databases </title>
    <year> 1995 </year>
  </book>
  <book price="55">
    <publisher> Freeman </publisher>
    <author> Jeffrey D. Ullman </author>
    <title> Principles of Database and Knowledge Base Systems </title>
    <year> 1998 </year>
  </book>
</bib>
```

Basic FLWR

Find all book titles published after 1995:

```xml
<bib> { 
for $x in doc("bib.xml")/bib/book
where $x/year/text() > 1995
return $x/title
} </bib>
```

Result:

```xml
<bib><title> Principles of Database and Knowledge Base Systems</title></bib>
```
FLWR vs. XPath expressions

Equivalently

```
for $x in doc("bib.xml")/bib/book[year/text() > 1995]/title
return $x
```

And even shorter:

```
```

---

Result Structuring

- Find all book titles and the year when they were published:

```
for $x in doc("bib.xml") /bib/book
return <answer>
    <title>{$x/title/text() } </title>
    <year>{$x/year/text() } </year>
</answer>
```

Braces {  } denote evaluation of enclosed expression
Result Structuring

- Notice the use of "{ " and " }"
- What is the result without them?

```xml
for $x in doc("bib.xml")/bib/book
return <answer>
  <title> $x/title/text() </title>
  <year> $x/year/text() </year>
</answer>
```

XQuery Joins and Nesting

For each author of a book by Addison-Wesley, list all books she published:

```xml
for $b in doc("bib.xml")/bib,
  $a in $b/book[publisher/text()="Addison-Wesley"]/author
return <result>
  { $a,
    for $t in $b/book[author/text()=$a/text()]/title
    return $t
  }
</result>
```

In the `return` clause comma concatenates XML fragments
XQuery Nesting

Result:

```
<result>
    <author>Jones</author>
    <title> abc </title>
    <title> def </title>
</result>
<result>
    <author>Smith</author>
    <title> ghi </title>
</result>
```

Aggregates

Find all books with more than 3 authors:

```
for $x in doc("bib.xml")/bib/book 
  where count($x/author)>3
return $x
```

- `count` = a function that counts
- `avg` = computes the average
- `sum` = computes the sum
- `distinct-values` = eliminates duplicates
Aggregates

Same thing:

```xml
for $x in doc("bib.xml")/bib/book[count(author)>3]
return $x
```

Print all authors who published more than 3 books – be aware of duplicates!

```xml
for $b in doc("bib.xml")/bib,
    $a in distinct-values($b/book/author/text())
where count($b/book[author/text()=$a])>3
return <author> { $a } </author>
```
Aggregates

Find books whose price is larger than average:

```xml
for $b in doc("bib.xml")/bib
let $a:=avg($b/book/price/text())
for $x in $b/book
where $x/price/text() > $a
return $x
```

Result Structure

“Flatten” the authors, i.e. return a list of (author, title) pairs

```xml
for $b in doc("bib.xml")/bib/book,
   $x in $b/title/text(),
   $y in $b/author/text()
return <answer>
   <title> { $x } </title>
   <author> { $y } </author>
</answer>
```

Result:
```
<answer>
   <title> abc </title>
   <author> efg </author>
</answer>
<answer>
   <title> abc </title>
   <author> hkj </author>
</answer>
```
Result Structure

For each author, return all titles of her/his books

for $b$ in doc("bib.xml")/bib, $x$ in $b$/book/author/text()
return
<answer>
  <author> $x$ </author>
  { for $y$ in $b$/book[author/text()=$x]/title
    return $y } 
</answer>

Result:
<answer>
  <author> efg </author>
  <title> abc </title>
  <title> klm </title>
  . . . .
</answer>

What about duplicate authors?

Result Structure

Eliminate duplicates:

for $b$ in doc("bib.xml")/bib,
  $x$ in distinct-values($b$/book/author/text())
return
<answer>
  <author> $x$ </author>
  { for $y$ in $b$/book[author/text()=$x]/title
    return $y } 
</answer>
SQL and XQuery Side-by-side

Product(pid, name, maker)
Company(cid, name, city)

Find all products made in Seattle

SELECT x.name
FROM Product x, Company y
WHERE x.maker=y.cid
    and y.city="Seattle"

for $r in doc("db.xml")/db,
   $x in $r/Product/row,
   $y in $r/Company/row
where
   $x/maker/text()=$y/cid/text()
   and $y/city/text() = "Seattle"
return { $x/name }
XQuery Variables

- **for $x in expr** -- binds $x to each value in the list expr

- **let $x := expr** -- binds $x to the entire list expr
  - Useful for common sub-expressions and for aggregations

XQuery: LET

Find all publishers that published more than 100 books:

```xml
<big_publishers>
{   for $p in distinct-values(//publisher/text())
    let $b := /db/book[publisher/text() = $p]
    where count($b) > 100
    return <publisher> { $p } </publisher>
}
</big_publishers>
```

$b is a collection of elements, not a single element

`count` = a (aggregate) function that returns the number of elms
FOR vs. LET

FOR
• Binds *node variables* → iteration

LET
• Binds *collection variables* → one value

FOR

```xml
for $x$ in /bib/book
return <result> { $x } </result>
```

Returns:
```xml
<result> <book>...</book></result>
<result> <book>...</book></result>
<result> <book>...</book></result>
...
```

LET

```xml
let $x$ := /bib/book
return <result> { $x } </result>
```

Returns:
```xml
<result> <book>...</book>
<book>...</book>
<book>...</book>
...
</result>
```
Collections in XQuery

- Ordered and unordered collections
  - `/bib/book/author/text()` = an ordered collection: result is in document order
  - `distinct-values(/bib/book/author/text())` = an unordered collection: the output order is implementation dependent
- let $a := /bib/book$ → $a$ is a collection
- $b/author$ → a collection (several authors...)

```xquery
return <result> { $b/author } </result>
```

SQL and XQuery Side-by-side

Product(pid, name, maker, price) → Find all product names, prices, sort by price

```sql
SELECT x.name, x.price
FROM Product x
ORDER BY x.price
```

```xquery
for $x in doc("db.xml")/db/Product/row
order by $x/price/text()
return <answer>
    { $x/name, $x/price }
</answer>
```
XQuery’s Answer

<answer>
  <name> abc </name>
  <price> 7 </price>
</answer>
<answer>
  <name> def </name>
  <price> 23 </price>
</answer>

Notice: this is NOT a well-formed document!
(Why ???)

Producing a Well-Formed Answer

<myQuery>
{   for $x in doc("db.xml")/db/Product/row
    order by $x/price/text()
    return <answer>
        { $x/name, $x/price }
    </answer>
}
</myQuery>
XQuery’s Answer

<myQuery>
  <answer>
    <name> abc </name>
    <price> 7 </price>
  </answer>
  <answer>
    <name> def </name>
    <price> 23 </price>
  </answer>
  . . . .
</myQuery>

Now it is well-formed!

SQL and XQuery Side-by-side

For each company with revenues < 1M count the products over $100

```
SELECT y.name, count(*)
FROM Product x, Company y
WHERE x.price > 100 and x.maker=y.cid and y.revenue < 1000000
GROUP BY y.cid, y.name
```

```xml
for $r in doc("db.xml")/db,
  $y in $r/Company/row[revenue/text()<1000000]
return
  <proudCompany>
    <companyName> { $y/name/text() } </companyName>
    <numberOfExpensiveProducts>
      { count($r/Product/row[maker/text()=$y/cid/text()][price/text()>100]) }
    </numberOfExpensiveProducts>
  </proudCompany>
```
SQL and XQuery Side-by-side

Find companies with at least 30 products, and their average price

SELECT y.name, avg(x.price)
FROM Product x, Company y
WHERE x.maker = y.cid
GROUP BY y.cid, y.name
HAVING count(*) > 30

An element

for $r in doc("db.xml")/db,
  $y in $r/Company/row
let $P := $r/Product/row[maker/text()=$y/cid/text()]
where count($P) > 30
return
  <theCompany>
    <companyName> { $y/name/text() } </companyName>
    <avgPrice> avg($P/price/text()) </avgPrice>
  </theCompany>

XQuery

Summary:
• FOR-LET-WHERE-RETURN = FLWR
Practical Example: Galax

$ more iis0.xq
<bib> {
for $x$ in doc("bib.xml")/bib/book/author[first-name]
return <result> {$x} </result>
} </bib>

$ galax-run iis0.xq
<bib><result><author><first-name>Rick</first-name>
    <last-name>Hull</last-name>
</author></result></bib>

http://www.galaxquery.org

Conclusion

- **XML parsers** are required for detailed usage of XML encoded data
- **XPath** provides a simple query language
- **XQuery** is an enhanced version of XPath (SQL like)