XML Technologies and Applications

Rajshekhar Sunderraman

Department of Computer Science
Georgia State University
Atlanta, GA 30302
raj@cs.gsu.edu

V (a). XML Querying: XPath

December 2005

Outline

- Introduction
- XML Basics
- XML Structural Constraint Specification
  - Document Type Definitions (DTDs)
  - XML Schema
- XML/Database Mappings
- XML Parsing APIs
  - Simple API for XML (SAX)
  - Document Object Model (DOM)
- XML Querying and Transformation
  - XPath
  - XSLT
  - XQuery
- XML Applications

XML Query Languages

- XPath – core query language. Very limited, a glorified selection operator. Very useful, though: used in XML Schema, XSLT, XQuery, many other XML standards
- XSLT – a functional style document transformation language. Very powerful, very complicated
- XQuery – W3C standard. Very powerful, fairly intuitive, SQL-style
- SQL/XML – attempt to marry SQL and XML, part of SQL:2003

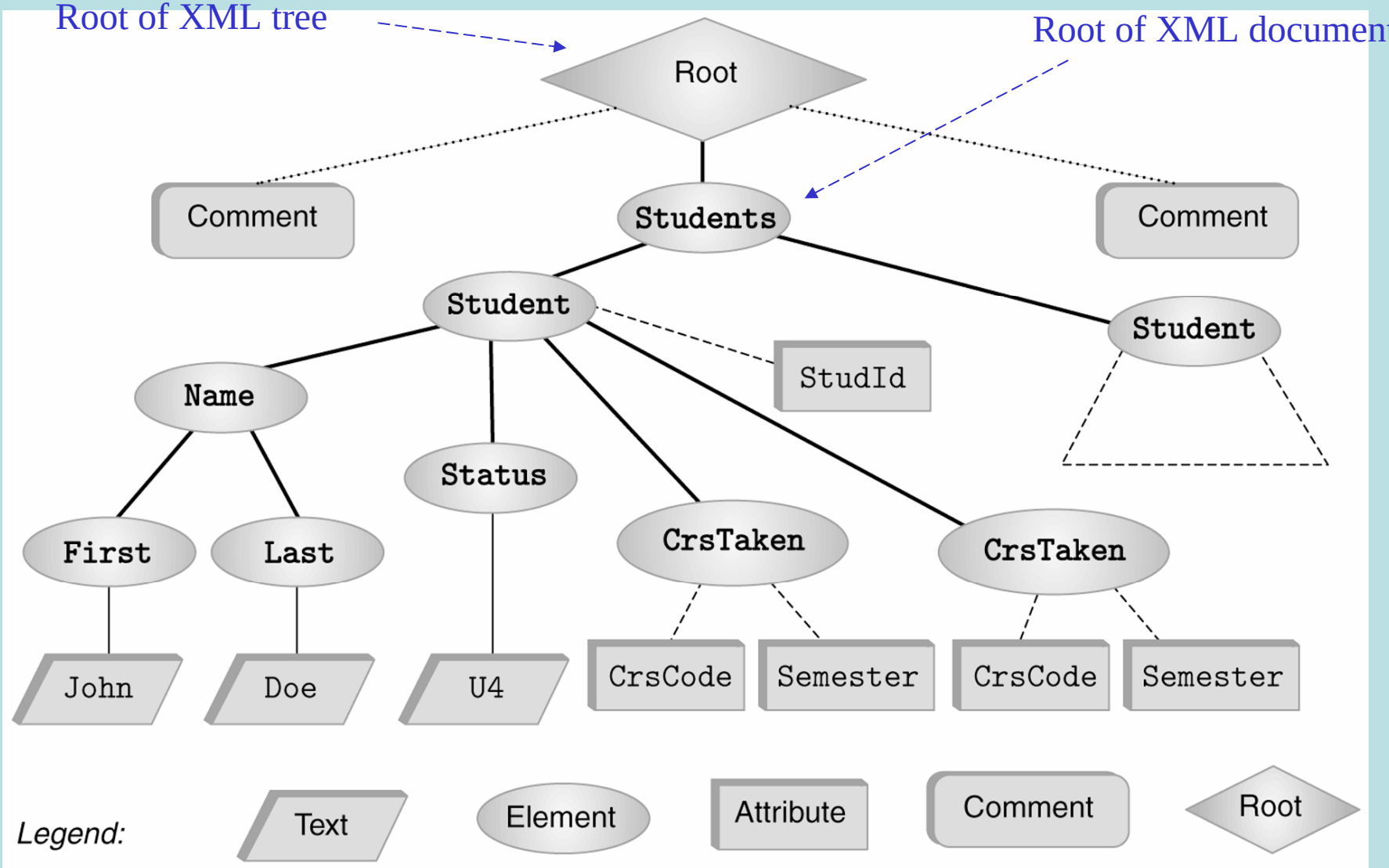
Why Query XML?

- Need to extract parts of XML documents
- Need to transform documents into different forms
  - Another XML form
  - HTML (to display on a Web browser)
  - Other (e.g. bibtex)
- Need to relate – join – parts of the same or different documents

XPath

- Analogous to path expressions in object-oriented languages (e.g., OQL) or file specification in UNIX
- Extends path expressions with query facility
- XPath views an XML document as a tree
  - Root of the tree is a *new* node, which doesn't correspond to anything in the document
  - Internal nodes are elements
  - Leaves are either
    - Attributes
    - Text nodes
    - Comments

XPath Document Tree



Sample Document Corresponding to the Tree

```
<?xml  version="1.0" ?>
<!-- Some  comment -->
<Students>
  <Student StudId="111111111" >
    <Name><First>John</First><Last>Doe</Last></Name>
    <Status>U2</Status>
    <CrsTaken  CrsCode="CS308"  Semester="F1997" />
    <CrsTaken  CrsCode="MAT123"  Semester="F1997" />
  </Student>
  <Student StudId="987654321" >
    <Name><First>Bart</First><Last>Simpson</Last></Name>
    <Status>U4</Status>
    <CrsTaken  CrsCode="CS308"  Semester="F1994" />
  </Student>
   ...
</Students>
<!-- Some other comment -->
```

Terminology

- *Parentl child* nodes, as usual
- Child nodes (that are of interest to us) are of types *text*, *element*, *attribute*
  - We call them *t-children*, *e-children*, *a-children*
  - Also, *et-children* are child-nodes that are either elements or text, *ea-children* are child nodes that are either elements or attributes, etc.
- Ancestor/descendant nodes – as usual in trees

XPath Basics

- An XPath expression takes a document tree as input and returns a set of nodes of the tree
- Expressions that *start* with **/** are *absolute path expressions*
  - Expression **/** – returns root node of XPath tree
  - **/Students/Student** – returns all Student-elements that are children of Students elements, which in turn must be children of the root
  - **/Student** – returns empty set (no such children at root)

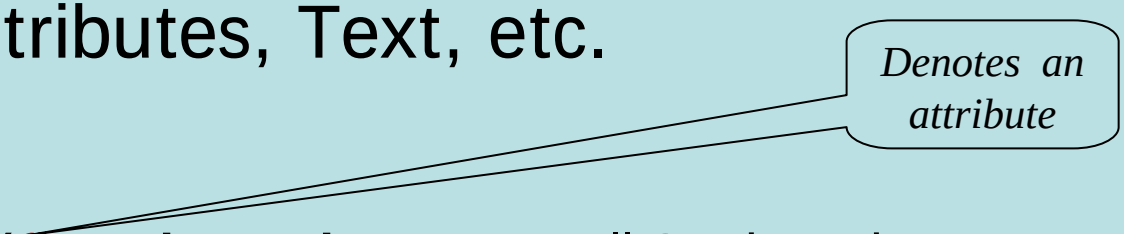
XPath Basics (cont'd)

- *Current* (or *context* node) – exists during the evaluation of XPath expressions (and in other XML query languages)

- **.** – denotes the current node; **..** – denotes the parent
  - **`foo/bar`** – returns all bar-elements that are children of foo nodes, which in turn are children of the current node
  - **`./foo/bar`** – same
  - **`../abc/cde`** – all cde e-children of abc e-children of the parent of the current node

- Expressions that don't start with **/** are *relative* (to the current node)

Attributes, Text, etc.



- **/Students/Student/@StudentId** – returns all StudentId

  a-children of Student, which are e-children of Students, which are children of the root

- /Students/Student/Name/Last/text( ) – returns all t-children of Last e-children of ...

- /comment( ) – returns comment nodes under root

- XPath provides means to select other document components as well

Overall Idea and Semantics

- An XPath expression is:
  locationStep1/locationStep2/...

- *Location step*:
  Axis::nodeSelector[predicate]

This is called ***full*** syntax.
We used ***abbreviated*** syntax before.
Full syntax is better for describing meaning. Abbreviated syntax is better for programming.

- Navigation *axis*:
  - *child, parent* – have seen
  - *ancestor, descendant, ancestor-or-self, descendant-or-self* , right-sibling,left-sibling etc.
  - some other

- *Node selector*: node name or wildcard; e.g.,
  - ./child::Student (we used ./Student, which is an abbreviation)
  - ./child::* – any e-child (abbreviation: ./*)

- *Predicate*: a selection condition; e.g.,
  Students/Student[CourseTaken/@CrsCode = “CS532”]

XPath Semantics

The meaning of the expression locationStep1/locationStep2/... is the set of all document nodes obtained as follows:

- Find all nodes reachable by locationStep1 from the current node
- For each node *N* in the result, find all nodes reachable from *N* by locationStep2; take the union of all these nodes
- For each node in the result, find all nodes reachable by locationStep3, etc.
- The value of the path expression on a document is the set of all document nodes found after processing the last location step in the expression

Overall Idea of the Semantics (Cont'd)

- locationStep1/locationStep2/... means:
  - Find all nodes specified by locationStep1
  - For each such node N:
    - Find all nodes specified by locationStep2 using N as the current node
    - Take union
  - For each node returned by locationStep2 do the same

- locationStep = axis::node[predicate]
  - Find all nodes specified by axis::node
  - Select only those that satisfy predicate

More on Navigation Primitives

- 2nd CrsTaken child of 1st Student child of Students:

  /Students/Student[1]/CrsTaken[2]

- All last CourseTaken elements within each Student element:

  /Students/Student/CrsTaken[last( )]

Wildcards

- Wildcards are useful when the exact structure of document is not known

- *Descendant-or-self* axis, **//** : allows to descend down any number of levels (including 0)
  - //CrsTaken – all CrsTaken nodes under the root
  - Students//@Name – all Name attribute nodes under the elements Students, who are children of the current node
  - *Note*:
    - ./Last and Last are same
    - .//Last and //Last are <u>different</u>

- The * wildcard:
  - * – any element: Student/*/text()
  - @* – any attribute: Students//@*

XPath Queries (selection predicates)

- Recall: Location step = Axis::nodeSelector[predicate]
- Predicate:
  - XPath expression = const | built-in function | XPath expression
  - XPath expression
  - built-in predicate
  - a Boolean combination thereof
- Axis::nodeSelector[predicate] $\subseteq$ Axis::nodeSelector but contains only the nodes that satisfy predicate
- Built-in predicate: special predicates for string matching, set manipulation, etc.
- Built-in function: large assortment of functions for string manipulation, aggregation, etc.

XPath Queries – Examples

- Students who have taken CS532:

  //Student[CrsTaken/@CrsCode="CS532"]

  True if : "CS532" ∈ //Student/CrsTaken/@CrsCode

- Complex example:

  //Student[Status="U3" and starts-with(.//Last, "A")
  and contains(concat(.//@CrsCode), "ESE")
  and not(.//Last = .//First)]

- Aggregation: sum( ), count( )

  //Student[sum(.//@Grade) div count(.//@Grade) > 3.5]

Xpath Queries (cont'd)

- Testing whether a subnode exists:
  - //Student[CrsTaken/@Grade] – students who have a grade (for some course)
  - //Student[Name/First or CrsTaken/@Semester or Status/text() = "U4"] – students who have either a first name or have taken a course in some semester or have status U4

- Union operator, | :

  //CrsTaken[@Semester="F2001"] | //Class[Semester="F1990"]

  – union lets us define *heterogeneous* collections of nodes

XPointer

- XPointer = URL + XPath

- Syntax:

  url # xpointer (XPathExpr1) xpointer (XPathExpr2) ...

  - Follow *url*
  - Compute XPathExpr1
    - Result non-empty? – return result
    - Else: compute XPathExpr2; and so on

- Example: you might click on a link and run a query against your Registrar's database

  http://yours.edu/Report.xml#xpointer(
  //Student[CrsTaken/@CrsCode="CS532"
  and CrsTaken/@Semester="S2002"])