

# XML Transformation

Ramesh Sankaranarayanan

Department of Computer Science  
Australian National University

COMP3410 IT in E-Commerce

## Outline

- 1 Introduction
  - XSL
- 2 XPath
  - Introduction
  - Syntax
  - Examples

## Outline

- 1 Introduction
  - XSL
- 2 XPath
  - Introduction
  - Syntax
  - Examples

## Transforming XML Documents - XSL

### Extensible Stylesheet Language (XSL)

XSL is a styling language. Used to both **transform** and **render** XML documents. Version 1.0 became a W3C recommendation in October, 2001. Version 1.1 became a recommendation in December, 2006. It consists of two parts:

- **XSL Transformations (XSLT)** to transform XML documents
- **XML Formatting Objects (FO)** for rendering XML documents

## XSLT

- Language for transforming XML documents
- Gives XML a lot of its power and flexibility
- Enables interoperability
- Uses **XPath** to access parts of an XML document
- Version 1.0 became a recommendation in November, 1999.  
Version 2.0 became a recommendation in January, 2007.

- 1 Introduction
  - XSL
- 2 XPath
  - Introduction
  - Syntax
  - Examples

## XML Path Language (XPath)

- Used to access parts of an XML document
- Designed to be used by a host language such as XSLT or XQuery.
- Provides basic facilities for manipulation of strings, numbers and booleans
- Has a natural subset that can be used for pattern matching. This feature is used in XSLT
- Version 1.0 became a recommendation in November, 1999.  
Version 2.0 became a recommendation in January, 2007.

## XPath

- Operates on the logical structure of an XML document
- Models an XML document as a tree of nodes
- Different type of nodes including element nodes, attribute nodes and text nodes
- Defines a way to compute a string-value for each type of node
- Fully supports XML Namespaces

## XPath Syntax

## XPath Data Model

- Views a document as a tree of **nodes**
- The tree has a **root**, which is different from the root element. Holds the document element and and comments/processing instructions that precede/follow it
- A node can have a **parent** node, **child** nodes, **ancestor** nodes and **descendant** nodes
- Element/Attribute** nodes correspond to elements/attributes in the XML document
- Likewise, there are **comment** and **processing instruction** nodes
- Text nodes contain the character content of the element and its descendants. Does not include attribute values, processing instructions or comments

## XPath Syntax

## Location Paths

- An instance of the XPath language is called an **expression**
- A **location path** is the most important type of expression
- A location path has a starting point called its **context node**
- Purpose is to select nodes from the document relative to the initial context node

## Location Paths - Examples

- `/` refers to the root node
- `/catalogue` selects the catalogue element node
- `/catalogue/book` selects the book element of catalogue. Each `/` steps you down the tree

## XPath Syntax

## Predicates

- Used to select nodes with a lot more flexibility, rather than just its type name
- Returns a set of locations. Can be thought of as pointers into the node tree expression

## Predicates - Examples

- `/catalogue/book[2]` refers to the second book of catalogue
- `book[title='The Blues Book']` matches book element with title **The Blues Book**
- `//book[@type="mystery"]` selects all books with value of the type attribute being **mystery**

## XPath Syntax

## Axes

- Can step through the tree in different directions, called **axes**
- Child axis (`/`), parent axis (`..`), self axis (`.`), descendant axis (`//`), ancestor axis, attribute axis (`@`), namespace axis and content axis are examples.
- Node tests and functions calls are available for use with location paths

## Axes - Examples

- `*` refers to any node of type element, attribute or namespace
- `node()` refers to any node
- `text()/comment()` refers to any text/comment node
- `processing-instruction()` refers to any processing instruction node

## XPath Syntax

## Unabbreviated syntax

- Have been using abbreviated syntax so far. the unabbreviated, precise syntax is more complicated
- An example is `/child::catalogue/child::book[position()=2]`
- The ancestor axis has no abbreviation. `ancestor::book` refers to ancestors of the context node that are named `book`

## XPath Examples

## Some examples:

- `book` book element nodes of the context node
- `//author` all author elements
- `//book/title` title elements that are children of all book elements
- `/catalog/*/*publisher` all publisher elements that are grandchildren of catalog
- `/*/*title` all title elements with two ancestors
- `//*` all elements in the document
- `../*` all descendants of the context node
- `..@*` attribute nodes attached to the parent or attached nodes of the context node

## XPath Examples

## Some examples:

- `book[4]` fourth `book` element child of the context node
- `book[@rating='G']/title` grandchildren of context node with element `title` that have parent `book` elements with a `rating` attribute value of `G`
- `//book[@review]/title` elements of type `title` that are children of all `book` elements that have a `review` attribute
- `book/title | book/author` all title or author elements that have `book` as the parent, with `book` being a child element of the context node