Lecture Notes 15
Extensible Style Language (XSL) and XML DTD

Anand Tripathi

CSci 4131

Internet Programming

Topics

- XML Document Type Definition (DTD)
- Introduction to Extensible Style Language (XSL)
  - XSL Transformation (XSLT)
  - It translates an XML document to another form
  - We will use it translate an XML document into an XHTML document for desired presentation.
  - XSL makes use of XPath query language
    - Query language to access DOM elements

XML DTD

- Developed prior to XML Schema.
- Many XML documents use DTDs.

Example:

```
<!ELEMENT  book (title, author, publisher, edition, year)  >
<!ELEMENT  title (#PCDATA)  >
<!ELEMENT  author (#PCDATA)  >
<!ELEMENT  publisher (#PCDATA)  >
<!ELEMENT  edition (#PCDATA)  >
```

PCDATA represents parsed character data

XML DTD

Suppose that we want to further refined the structure of the author element with child elements for firstname and lastname:

```
<!ELEMENT  author (firstname, lastname)  >
<!ELEMENT  firstname (#PCDATA)  >
<!ELEMENT  lastname (#PCDATA)  >
```

XML DTD

We want have

- At least one or more *author* elements for a book
- Zero or one *year* element
- Zero or one *edition* element
- Any number of reader reviews

Example:

```
<!ELEMENT  book (title, author+, publisher, edition?,
    year?,   reviews*)  >
```

Empty Element

```
<!ELEMENT  LogoImage  EMPTY >
```

This element may have some attributes.

```
<LogoImage file=“image-file-path” />
```

Defining Choices for Child Elements

```
<!ELEMENT book (title, author, publisher,
    hardcover|softcover|eBook) >
<!ELEMENT hardcover EMPTY >
<!ELEMENT softcover EMPTY >
<!ELEMENT eBook EMPTY >
```

Attributes

```
<!ELEMENT  book (title, author, publisher, edition, year)  >
<!ATTLIST  book medium  CDATA >
```

OR

`<!ATTLIST  book medium  CDATA #IMPLIED >` means optional

```
<!ATTLIST  book medium  CDATA #REQUIRED >
```

Default Values for Attributes

```
<!ELEMENT  book (title, author, publisher, edition, year)  >
<!ATTLIST  book medium  CDATA “paper” >
```

Limiting the possible values for an attribute:

```
<!ATTLIST  book medium   ( paper|CD|web )  #REQUIRED>
```

Attributes with Unique Values

- Adding ID attribute to an element

```
<!ELEMENT  book (title, author, publisher, edition, year)  >
<!ATTLIST  book isbn  ID  #REQUIRED >

<!ELEMENT textbook (course, semester, year)  >
<!ATTLIST  catalogReference   IDREF  #REQUIRED >
```

Restricting Attribues to Valid XML Names

- <!ATTLIST someAttr NMTOKEN #REQUIRED>

- NMTOKEN can be valid XML token:
  - Begins with a letter, underscore or colon.
  - It may contain any number of letters, digits, and underscores

Internal DTD

```
<?xml version="1.0" ?>
<!DOCTYPE  book [
     <!ELEMENT book (title, author, publisher) >
     …..
  ]
<book>
….
</book>
```

Using an external DTD

```
<?xml  version=“1.0” ?>
<!DOCTYPE  book  SYSTEM “book.dtd” >
<book>
   ……
</book>
```

This declares “book” as the root element of the XML.

SYSTEM indicates that the DTD is available else where in the system.

A URI parameter, in this case book.dtd, indicates the location of the DTD.

Using a PUBLIC External DTD

<!DOCTYPE book PUBLIC "formal public identifier (FPI)" URI >

- This declares "book" as the root element of the XML.
- PUBLIC indicates that the DTD may be available in some public repository with its unique name give in FPI
- A URI parameter indicates the location of the DTD. This can be optional for public DTDs.

```
<!DOCTYPE  html   PUBLIC   "-//W3C//DTD HTML 4.01 Transitional//EN">

<!DOCTYPE  html   PUBLIC   "-//W3C//DTD HTML 4.01 Transitional//EN“
    http://www.w2.org/TR/xhtml1/DTD/xhtml1-transitional.dtd >
```

Formal Public Identifier (FPI)

FPI format has following fields

Type // Owner // Description // Language

-//W3C//DTD HTML 4.01 Transitional//EN

Type:

-	stands for DTD as "not a recognized standard"
+	DTD is approved non-ISO standard
ISO	DTD is ISO standard

XHTML 1.0 Version Declarations

Transitional Version: This is the most used version, XHTML 1.0. Corresponds to HTML 4.0.1. It has support for some elements that have been deprecated. Pages without using stylesheets.

```
<!DOCTYPE  html   PUBLIC   "-//W3C//DTD HTML 4.01 Transitional//EN“
    http://www.w2.org/TR/xhtml1/DTD/xhtml1-transitional.dtd >
```

XTML Strict: Subset of XHTML transitional. Supports use of CSS.

```
<!DOCTYPE  html   PUBLIC   "-//W3C//DTD XHTML 1.0 Strict//EN“
    http://www.w2.org/TR/xhtml1/DTD/xhtml1-strict.dtd >
```

XHTML Frameset: Use this when you want to use frames. It includes all elements of XHTML Transitional plus elements for defining frames.

```
<!DOCTYPE  html   PUBLIC   "-//W3C//DTD XHTML 1.0 Frameset//EN“
    http://www.w2.org/TR/xhtml1/DTD/xhtml1-frameset.dtd >
```

Well-formed vs Valid XHTML Documents

1. A well-formed document adheres to structuring rules, such as presence of both opening tag and closing tag.

   A well-formed document may include element tags that are not part of the XHTML specification.

2. A valid document is a well-formed, and moreover it also conforms to the rules of a schema or DTD.

   Validity is a stricter property.

Extensible Style Language (XSL)

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="myStyle.xsl" ?>
<book>
    <title> Introduction to XML </title>
    <author> IBM  </author>
    <description>  This is about XML  </description>
    <isbn>  123456 </isbn>
    <price>  19.99 </price>
    <publisher> IBM  </publisher>
    <edition>  Second </edition>
    <year> 2002 </year>
</book>
```

XML Document with Stylesheet

- This XML document contains one processing instruction which directs the use of a stylesheet in Extensible Stylesheet Language.
  - It specifies myStyle.xsl file to be used for stylesheet.
- Most browsers are equipped with appropriate XSL transformation programs to convert this XML into XHTML.
- Click here to see this XML transformed to XHTML by the browser

XSL Stylesheet File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0" >


 <xsl:output method = "html" omit-xml-declaration = "no"
     doctype-system = "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
     doctype-public = "-//W3C//DTD XHTML 1.0 Strict//EN"/>
```

… continued on the next page

```
  <xsl:template match="/">
 <html>  <head>  </head>
   <body>
     <h1>  Book Information </h1>
      The title of the book is <xsl:value-of select="book/title" />.
      <br/>
       It was written by <xsl:value-of select="book/author"/>.
      <br/>
       This book was published by <xsl:value-of select="book/publisher" />
        in the year  <xsl:value-of select="book/year" />.
       <p>  To read this book spend US-Dollars
         <xsl:value-of select="book/price" />
       </p>
   </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

XSL Stylefile

- It looks mostly as an XHTML document which contains directives to insert data extracted from the XML document.
- It defines rules for extracting information present under various elements of the XML documents and inserting
- <xsl:value-of select="book/title" />
- select specifies a patterns of elements to select one or more DOM node element names or attribute names
- value-of returns its string value
- These directives are XPath expressions to query the DOM tree of the XML document.
- It contains patterns to match element hierarchy.
- XSL document starts with matching the root element.

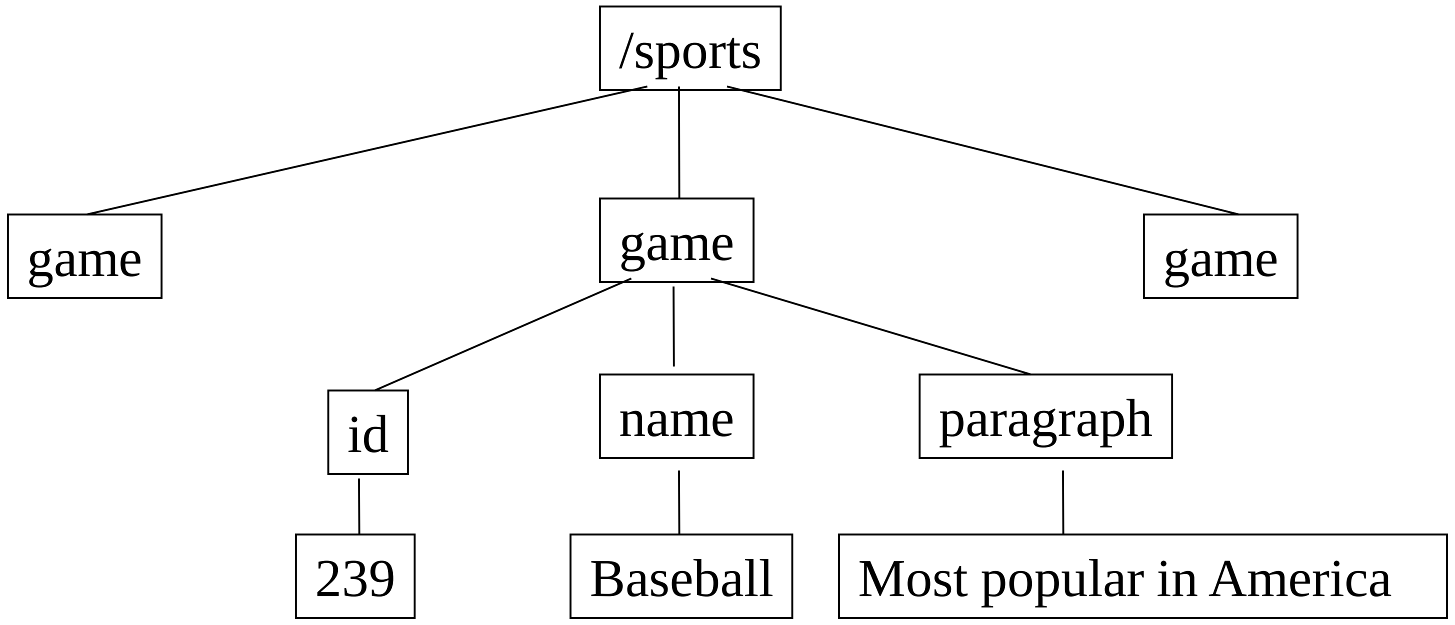
XML Document with Stylesheet

- This XML document contains one processing instruction which directs the use of a stylesheet in Extensible Stylesheet Language.
  - It specifies myStyle.xsl file to be used for stylesheet.
- Most browsers are equipped with appropriate XSL transformation programs to convert this XML into XHTML.
- Click here to see this XML transformed to XHTML by the browser

Example 1: XML Document (Page 538 of Deitel Book, 5th edition)

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl"   href = "sports.xsl"?>
<sports>
   <game id = "783">
      <name>Cricket</name>
      <paragraph> More popular among commonwealth nations.  </paragraph>
   </game>
   <game id = "239">
      <name>Baseball</name>
      <paragraph>  More popular in America  </paragraph>
   </game>
   <game id = "418">
      <name>Soccer (Futbol)</name>
      <paragraph>  Most popular sport in the world.  </paragraph>
   </game>
</sports>
```

DOM Tree


/sports
game
game
game
id
name
paragraph
239
Baseball
Most popular in America

XML Document with Stylesheet

- This XML document contains one processing instruction which directs the use of a stylesheet in Extensible Stylesheet Language.
  - It specifies sports.xsl file to be used for stylesheet.
- Most browsers are equipped with appropriate XSL transformation programs to convert this XML into XHTML.
- Click here to see this XML transformed to XHTML by the browser

Sports.xsl

```
<?xml version = "1.0"?>
<xsl:stylesheet version = "1.0" xmlns:xsl =
    "http://www.w3.org/1999/XSL/Transform">
  <xsl:output method = "html" omit-xml-declaration = "no"
    doctype-system =
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
    doctype-public = "-//W3C//DTD XHTML 1.0 Strict//EN"/>
  <xsl:template   match = "/">
  <!-- match root element -->
  <html xmlns = "http://www.w3.org/1999/xhtml">
    <head>
      <title>Sports</title>
    </head>
```

```
 <body>
        <table border = "1" bgcolor = "wheat">
           <thead>
              <tr>   <th>ID</th>     <th>Sport</th>    <th>Information</th>   </tr>
           </thead>
           <xsl:for-each select = "/sports/game">
              <tr>
                 <td>   <xsl:value-of select = "@id"/>       </td>
                 <td>   <xsl:value-of select = "name"/>     </td>
                 <td>   <xsl:value-of select = "paragraph"/>   </td>
              </tr>
           </xsl:for-each>
        </table>
     </body>
  </html>
  </xsl:template>
</xsl:stylesheet>
```

XPath Conditional Expression

This is example sports-v2.xml

```
<xsl:for-each select = "/sports/game">
            <xsl:if  test="name!='Cricket'" >
             <tr>
                <td><xsl:value-of select = "@id"/></td>
                <td><xsl:value-of select = "name"/></td>
                <td><xsl:value-of select = "paragraph"/></td>
             </tr>
            </xsl:if>
</xsl:for-each>
```

Click here to see the example

XPath

XPath

- XSLT makes use of XPath facilities.
- XPath provides primitives for searching and accessing DOM tree nodes.
- DOM tree nodes are accessed using a naming scheme similar to file paths.
- Seven types of nodes in the DOM tree:
  - Root node, element node, attribute node, text, comment node, processing instruction, XML-namespace declaration

XPath Primitives

- <xsl:value-of select=“expression”>

- It selects an XML element and returns its value, which is inserted into the output document.
- “expression” is an XPath expression, which are similar in structure to the filepath names:
  - Both absolute and relative paths

XML Document

```
<sports>
  <game id = "783">
    <name>Cricket</name>
    <paragraph>
      More popular among commonwealth nations.
    </paragraph>
  </game>
  <game id = "239">
    <name>Baseball</name>
    <paragraph>
      More popular in America.
    </paragraph>
  </game>
  <game id = "418">
    <name>Soccer</name>
    <paragraph>
      Most popular sport in the world.
    </paragraph>
  </game>
</sports>
```

XPath Expressions

This is example sports-v3.xml

Click here to see the example

- <xsl:value-of select ="/sports/game" />
- Result: Cricket More popular among commonwealth nations.
- <xsl:value-of select ="/sports/game/name" />
- Result: Cricket
- <xsl:value-of select ="/sports/game/@id"/>
- Result: 783
- <xsl:value-of select ="/sports/game/paragraph"/>
- Result: More popular among commonwealth nations.

…continued on the next page

XPath Expressions

- <xsl:value-of select ="/sports/game[@id='239']"/>
- Result: Baseball More popular in America.
- <xsl:value-of select ="/sports/game[@id='239']/paragraph"/>
- Result: More popular in America.
- <xsl:value-of select ="/sports/game[name='Soccer']"/>
- Result: Soccer Most popular sport in the world.
- <xsl:value-of select ="/sports/game[name='Soccer']/paragraph"/>
- Result: Most popular sport in the world.
- <xsl:value-of select ="/sports/game[name='Soccer']/name"/>
- Result: Soccer

XPathTemplate

```
<xsl:template  match=“pattern”>
  Some processing instruction, such as generate HTML code
</xsl:template>
```

- A template is conceptually similar to a function. If the current node satisfies the pattern, it applies processing instruction
- “match pattern” is an XPath expression, which are similar in structure to the filepath names:
  – Both absolute and relative paths, similar to file systems.
- Notion of “current node”
  – Processing instructions are applied to the current node, and the DOM tree under it, if it satisfies the pattern.

Difference between match and select

See this MSDN site for distinction between match and select

- Both are specified using Xpath expression.
- match checks if the current node satisfies the given pattern. If so, the enclosed process instructions are applied
- select identifies a node or set of nodes in the tree rooted at the current node, satisfying the given Xpath expression.
  - It identifies a subset of nodes for further processing
- select appears with elements with tags: value-of, for-each, apply-templates.
- match appears with element with tag template

XPathTemplate

```
<xsl:template  match=“/”>
  Some processing instruction, such as HTML code
</xsl:template>
```

- This pattern will apply the root node of the document. There is only one root node in any XML document.
- Root node will become the “current node”
- Processing instructions may select other nodes under the root by applying path matching and selection rules.
- A document may have multiple templates defined, but only the root template is automatically applied and executed.
  - All other templates have to be explicitly executed using apply-templates primitive.

```
<xsl:template match = "/"> <!-- match root element -->
  <html>
    <head>
      <title>Sports</title>
    </head>
    <body>
    List of Games.
    <br/>
    <xsl:apply-templates select="sports/game" />
    </body>
  </html>
  </xsl:template>
  <xsl:template match="game">
     <p>
     A. <xsl:value-of select ="name" />
     <br/>
     B. <xsl:value-of select ="@id"/>
     <br/>
     C. <xsl:value-of select ="paragraph"/>
     <br/>
     </p>
</xsl:template>
```

Click here to see this example.

XSL Elements

<xsl:for-each   select=“pattern” >

… this contains rules to be applied to each of the selected nodes

</xsl:for-each>

The XPath pattern is used to select a set of nodes.

The rules given between the matching for-each pairs are applied to each selected node and the subtree rooted at that node.

XSL Element: sort

<xsl:sort select=“pattern” data-type=“number|text” order=‘ascending|descending” / >

<xsl:sort select = "@number" data-type = "number” order = "ascending“ />

It is generally a child element of a xsl:for-each element or a template.

Before applying any rules, xsl:sort is used to sort the nodes in the ascending or descending order

XSL Element: output

- <xsl:oputput … />

Defines the various properties to be outputted as part of the transformed document.

For example XHTML DTD information for HTML documents.

- <xsl:variable name="varName" select="sum(…)" />

Defines a variable and assigne it a value

- <xsl:value-of select= "$varName" />

Conditional Processing of a Node

```
<xsl:if    test="expression>
   … conditional processing
</xsl>
```

Example:

```
 <xsl:for-each select = "/sports/game">
     <xsl:if  test="name!='Cricket'" >
               <tr>
                  <td><xsl:value-of select = "@id"/></td>
                  <td><xsl:value-of select = "name"/></td>
                  <td><xsl:value-of select = "paragraph"/></td>
               </tr>
     </xsl:if>
 </xsl:for-each>
```

Click here to see this example

Multiple Conditions

```
<xsl:choose>
 <xsl:when test=“expression1”>
     processing rules;
  </xsl:when>
   <xsl:when test=“expression2”>
     processing rules;
  </xsl:when>
  <xsl:otherwise>
     processing rules
  </xsl:otherise>
</xsl:choose>
```

Conditional Template Application

```
<xsl:template match="/">
<html xmlns = http://www.w3.org/1999/xhtml>
      <head>  <title>Sports</title>   </head>
      <body>
      List of Games.
      <br/>
          <xsl:apply-templates select="sports/game[@id!=239]" />
      </body>
   </html>
   </xsl:template>
   <xsl:template match="game">
       <p>  A. <xsl:value-of select ="name" /> <br/>
       B. <xsl:value-of select ="@id"/> <br/>
       C. <xsl:value-of select ="paragraph"/>  <br/> </p>
   </xsl:template>
```

Template Application Rules

The template application algorithm is as follows:

1. The first the root template is applied .
2. If the action part of the root template contains any apply-templates element, then a set of nodes matching the given path are selected.

   <apply-templates select="some-path" />

- The subtree under each selected node is traversed in depth-first search fashion as described next.
- A selected node becomes the "current-node".

Template application follows pre-order traversal of the tree at a node.

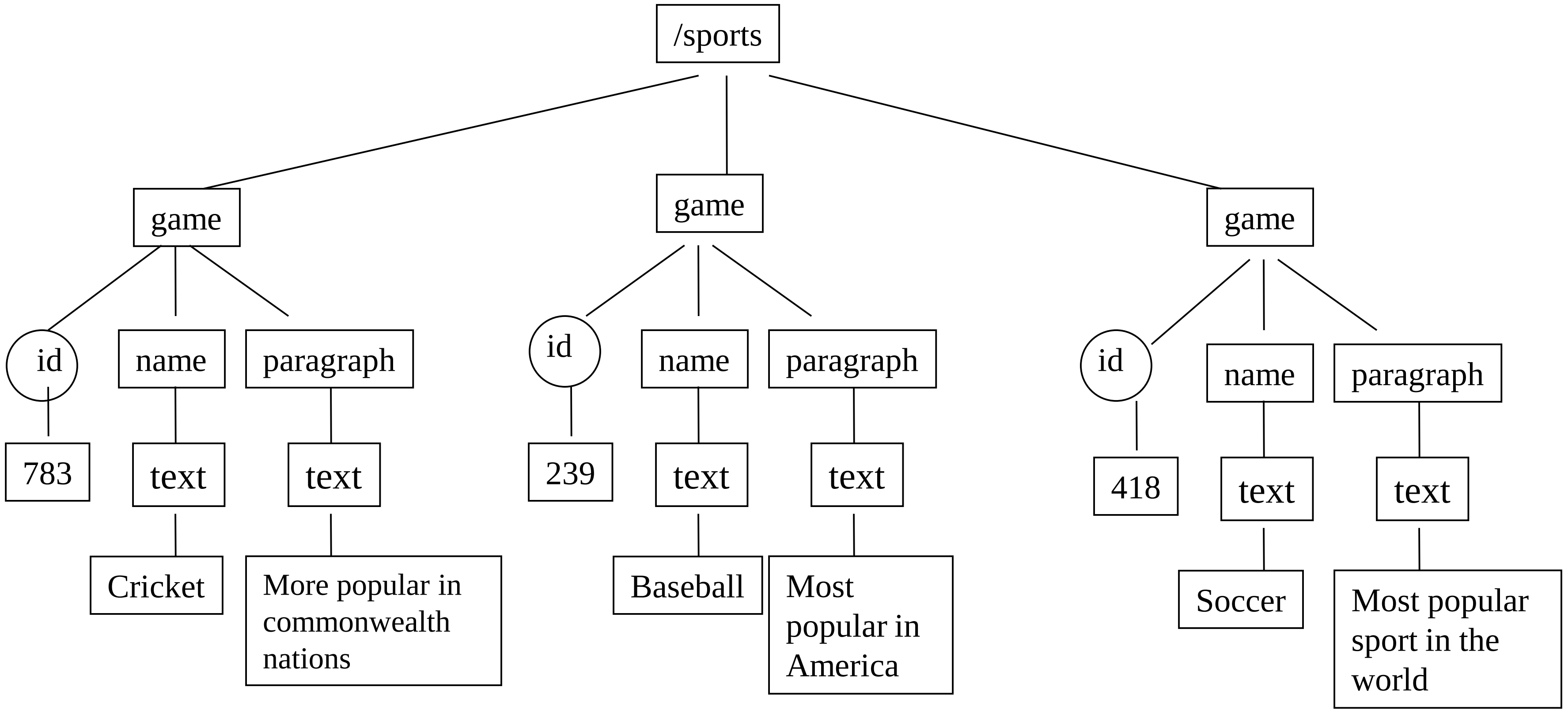
Template Application Rules

1. apply-templates command performs depth-first processing of the tree nodes at the "current node" to check if any of the templates are applicable at that node.
   - Most specific matching template is applied at that node
2. If any template matches, then the template action is executed and the traversal for the tree under that node is considered completed and the search continues with other nodes.
   - The body of the template is executed, which may generate some XHTML output or apply other DOM processing templates.
3. If no template matches, then
   - (a) Traversal continues with the next un-visited child node as the current node, and traversal continues with step (1)
   - (b) If all child nodes are visited, traversal backtracks to the parent node with step (3)
   - (c) If the current node is a leaf node, then the text-value of the node is output, the node is marked as visited, and the traversal backtracks to the parent node with step (b) above

Example 1: XML Document (Page 545 of Deitel Book)

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl"   href = "sports.xsl"?>
<sports>
   <game id = "783">
      <name>Cricket</name>
      <paragraph> More popular among commonwealth nations.  </paragraph>
   </game>
   <game id = "239">
      <name>Baseball</name>
      <paragraph>  More popular in America  </paragraph>
   </game>
   <game id = "418">
      <name>Soccer </name>
      <paragraph>  Most popular sport in the world.  </paragraph>
   </game>
</sports>
```

DOM Tree


/sports
game
game
game
id
name
paragraph
783
text
text
Cricket
More popular in commonwealth nations
id
name
paragraph
239
text
text
Baseball
Most popular in America
id
name
paragraph
418
text
text
Soccer
Most popular sport in the world

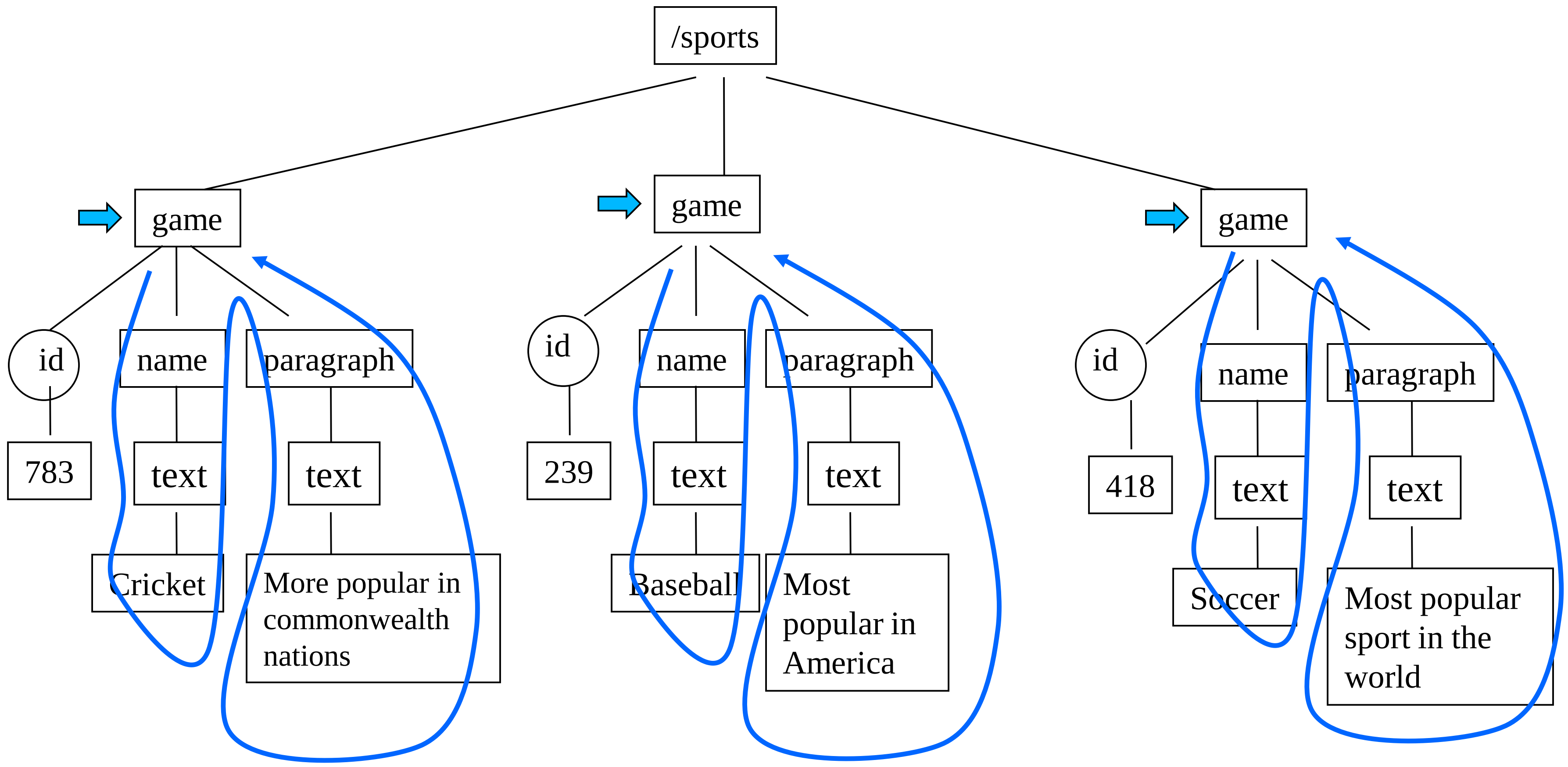
Example 1 of apply-templates

XSL file contains the following Xpath code:

```
<xsl:template match = "/"> <!-- match root element -->
    <html xmlns = "http://www.w3.org/1999/xhtml">
     <head>   <title>Sports</title>   </head>
     <body>
             List of Games    <br/>
           <xsl:apply-templates select="sports/game" />
     </body>
    </html>
  </xsl:template>

<! - -  Comment – No other templates are defined  - ->
```

Nodes selected for traversal



Example 1 - output

```
List of Games.
    Cricket More popular among commonwealth nations Baseball More
    popular in America Soccer Most popular sport in the world
```

(This example is in sports-v12.xml and sports-v12.xsl)

Example 2 of apply-templates

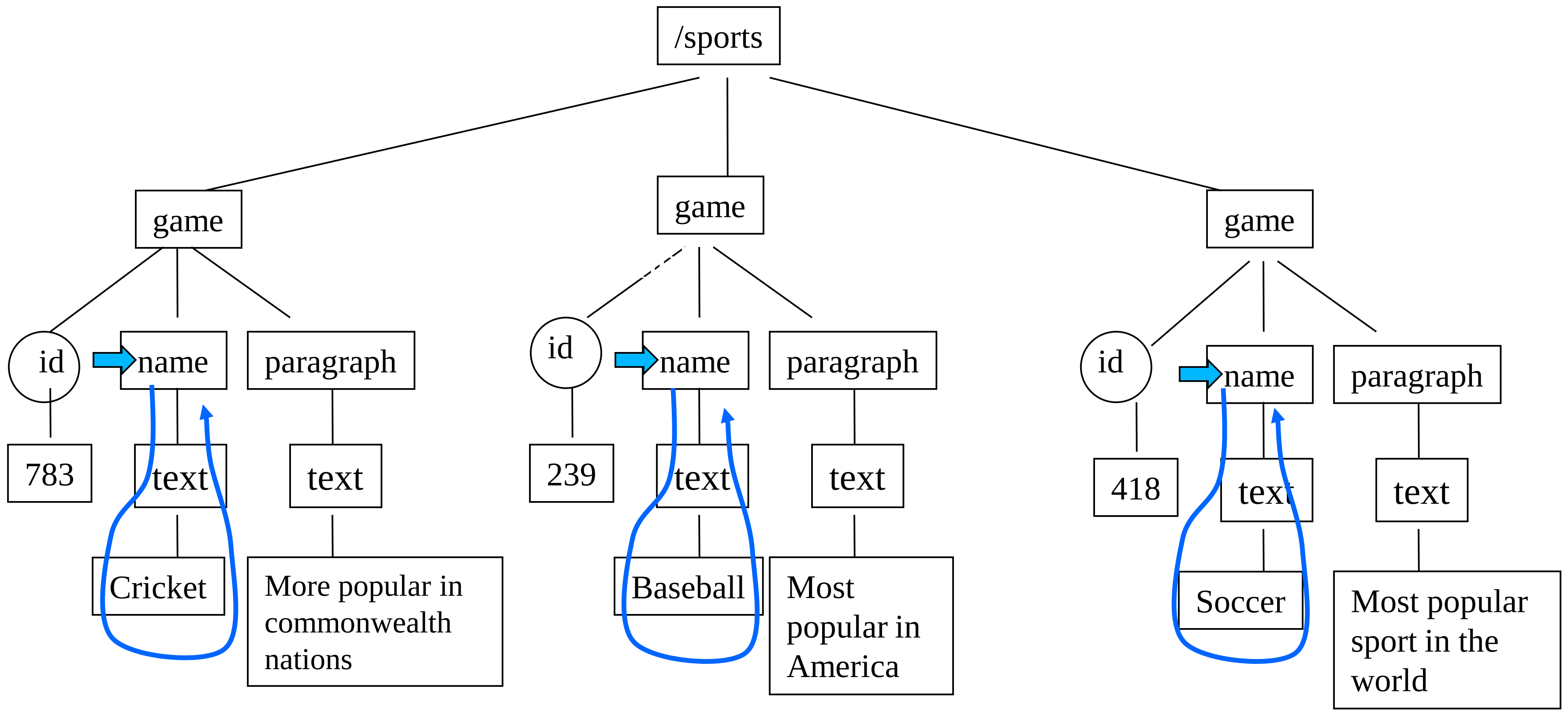
This example is given in sport-v13.xsl and sports-v13.xml

XSL file contains the following Xpath code:

```
<xsl:template match = "/"> <!-- match root element -->
     <html xmlns = "http://www.w3.org/1999/xhtml">
      <head>   <title>Sports</title>   </head>
      <body>
              List of Games    <br/>
            <xsl:apply-templates select="sports/game/name" />
      </body>
     </html>
   </xsl:template>

<! - -  Comment – No other templates are defined  - ->
```

Nodes selected for traversal



Example 2 - output

```
List of Games.
     Cricket  Baseball  Soccer
```

(This is example sports-v13.xml and sports-v13.xsl.)

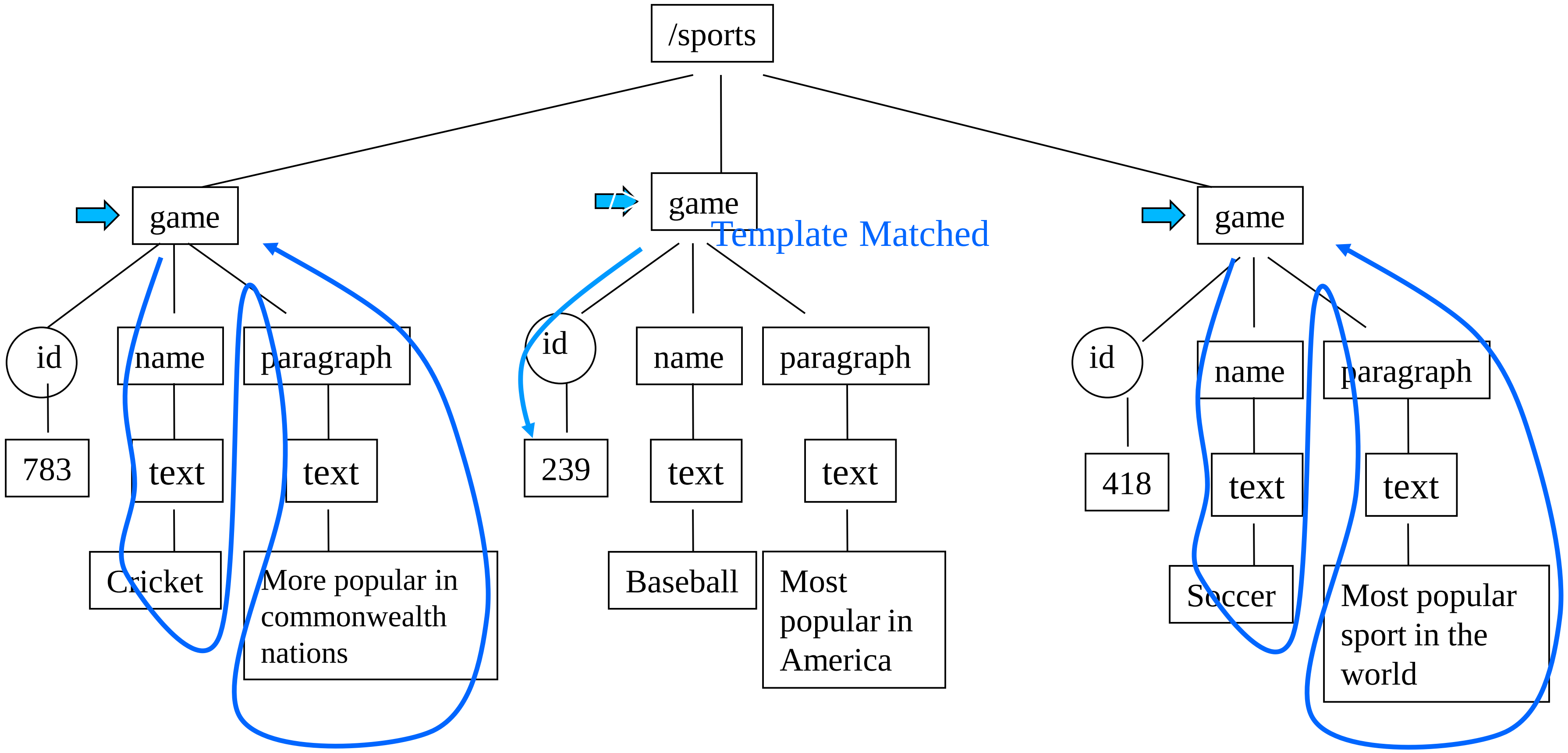
Example 3 of apply-templates

This example is in sports-v11.xsl and sports-v11.xml

XSL file contains the following Xpath code:

```
<xsl:template match = "/"> <!-- match root element -->
     <html xmlns = "http://www.w3.org/1999/xhtml">
      <head>   <title>Sports</title>   </head>
      <body>
              List of Games    <br/>
            <xsl:apply-templates select="sports/game" />
      </body>
     </html>
   </xsl:template>
 <xsl:template match="game[@id=239]">
     <br/> Found game with id=239  <br/>
  </xsl:template>
```

Nodes selected for traversal



Example 3 - output

```
List of Games.
    Cricket More popular among commonwealth nations
    Found game with id=239
    Soccer Most popular sport in the world
```

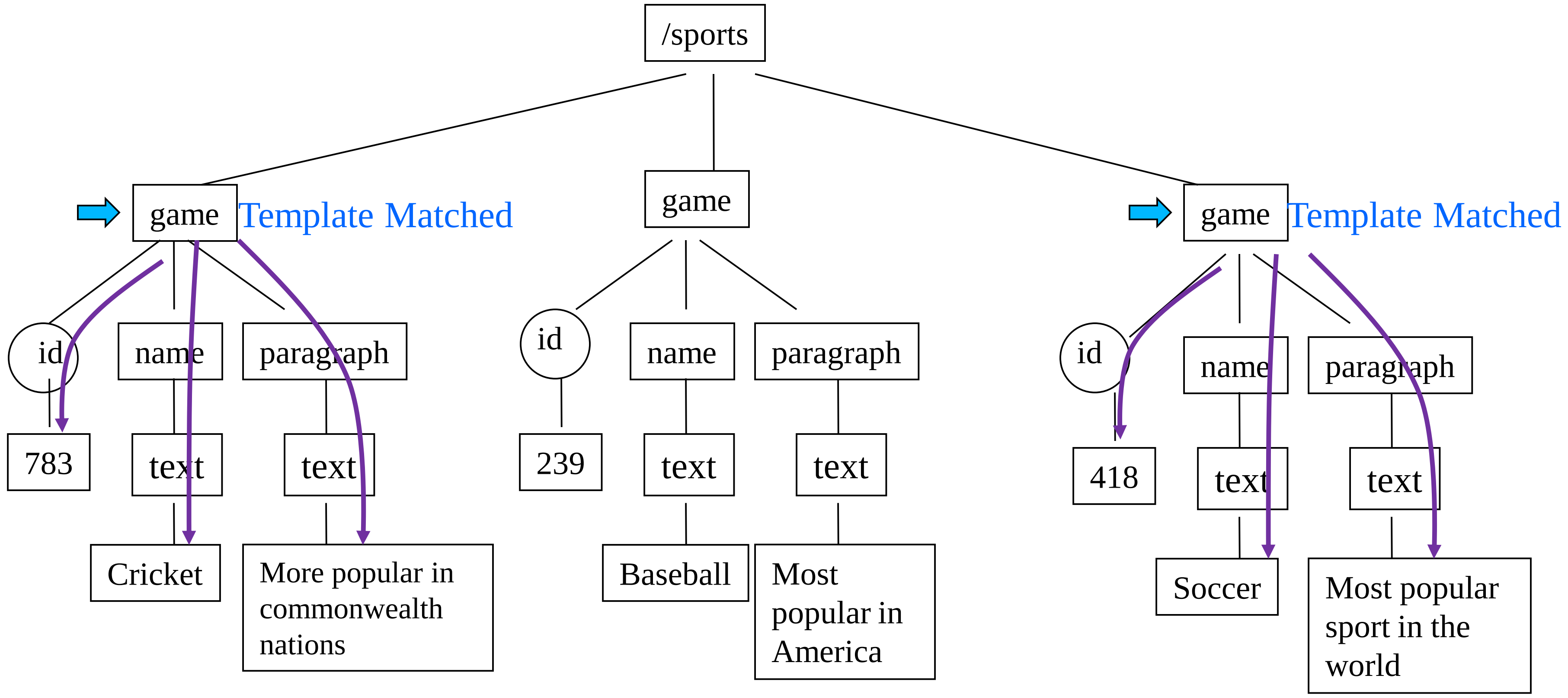
(This example is in sports-v11.xsl and sports-v11.xml.)

Conditional Template Application

sports-v5.xml and sport-v5.xsl

```
<xsl:template match = "/"> <!-- match root element -->
<html xmlns = "http://www.w3.org/1999/xhtml">
      <head>  <title>Sports</title>   </head>
      <body>
      List of Games.
      <br/>
          <xsl:apply-templates   select="sports/game[@id!=239]" />
      </body>
   </html>
   </xsl:template>
   <xsl:template match="game">
       <p>  A. <xsl:value-of     select ="name" /> <br/>
       B. <xsl:value-of select ="@id"/> <br/>
       C. <xsl:value-of select ="paragraph"/>  <br/> </p>
   </xsl:template>
```

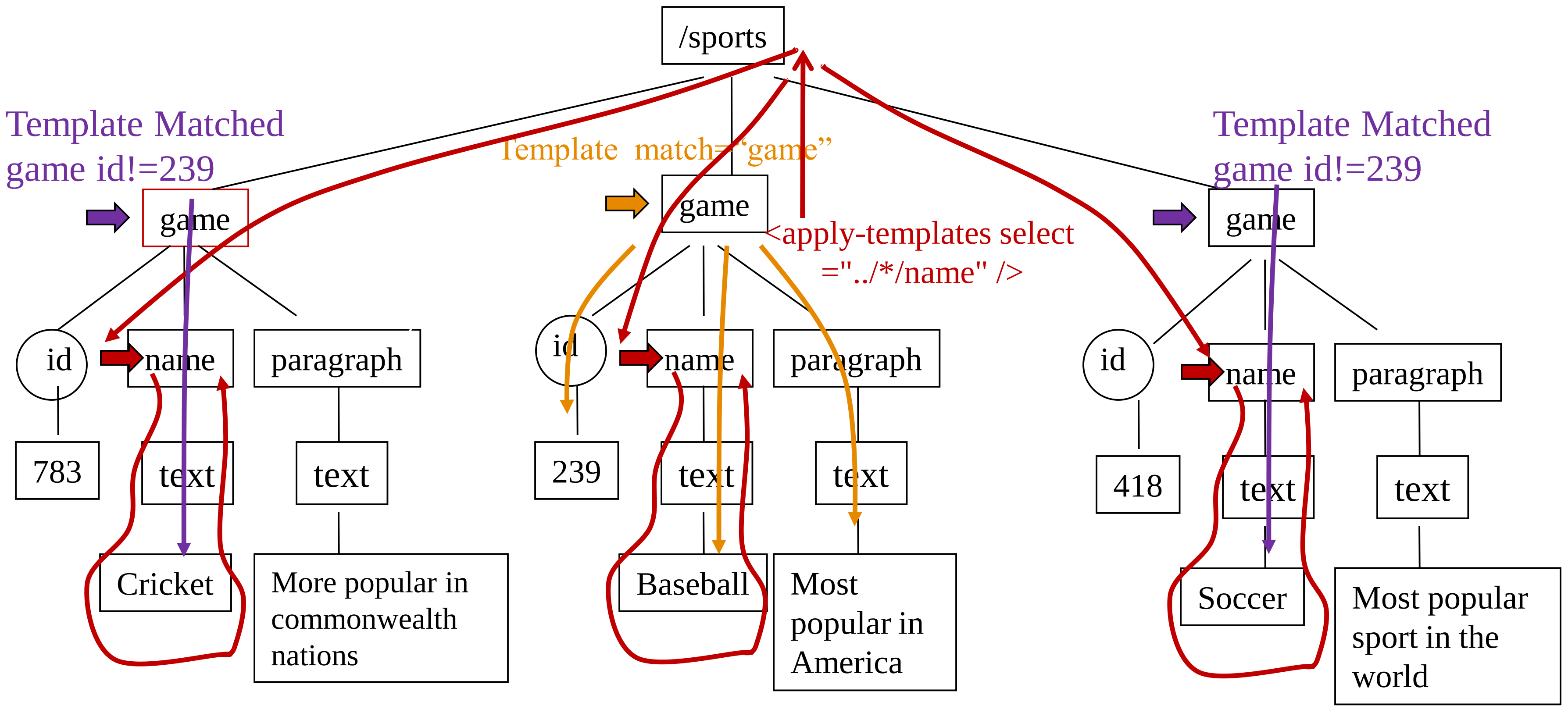
Nodes selected for traversal



Example 4 of apply-templates
(sports-v8.xml and sports-v8.xsl)

```
<xsl:template match = "/"> <!-- match root element -->
 <html xmlns = "http://www.w3.org/1999/xhtml">
     <body   List of Games. <br/>
    <xsl:apply-templates select="sports/game" />
    </body>  </html>
 </xsl:template>
 <xsl:template match="game[@id!=239]">
     SPORT: <xsl:value-of select =“name" />   <br/>
 </xsl:template>
 <xsl:template match="game">
     <p>  A. <xsl:value-of select ="name" />   <br/>
     B. <xsl:value-of select ="@id"/>  <br/>
     C. <xsl:value-of select ="paragraph"/>  <br/>
     D. Names of other games: <xsl:apply-templates select="../*/name" />   </p>
 </xsl:template>
```

Nodes selected for traversal



Example output

```
List of Games.
    SPORT: Cricket
    A. Baseball
    B. 239
    C. More popular in America.
    D. Names of other games: Cricket Baseball Soccer
SPORT: Soccer
```

(This is example sports-v8.xsl and sports-v8.xml.)

Example 5 of apply-templates

```
<BookList>
      <Book subject="Physics">
        <Title> Introduction to Physics </Title>
        <Author>   <first> Thomas </first>     <last> Gray </last>   </Author>
        <Author>   <first> William </first>   <last> Gray </last>    </Author>
        <Year>1994</Year>
      </Book>
      <Book subject="Chemistry">
        <Title> Introduction to Chemistry </Title>
        <Author>   <first> Johnson </first>    <last> Gray </last>   </Author>
        <Year>1990</Year>
      </Book>
</BookList>
```

Example 5 of apply-templates

```
<xsl:template match = "/">
    <xsl:apply-templates select="BookList/Book"/>
</xsl:template>

<xsl:template match="last">
</xsl:template>
```

What is the output produced by this XSL?

Example 5 of apply-templates

XML document will be displayed as follows:

Introduction to Physics Thomas William 1994 Introduction to Chemistry Johnson 1990

Example

See people.xml and people.xsl

```
<people>
<person>
          <name>
            <first> Jack     </first>
            <last> Jumping     </last>
          </name>
          <faculty  rank="Prof." />
          <department>
            <name> CS </name>
          </department>
</person>
```

```
<person>
            <name>
                  <first> Mickey </first>
                  <last> Mouse  </last>
            </name>
            <faculty  rank="Prof."> Assistant Professor </faculty>
            <department>
                  <name> CS </name>
            </department>
</person>
<person>
            <name>
                  <first> Larry </first>
                  <last> Limping </last>
            </name>
            <student rank="grad" />
            <department>
                  <name> CS </name>
            </department>
</person>
```

```
<person>
        <name>
            <first> Mocha </first>
            <last> Coffee </last>
        </name>
        <staff rank="manager" />
        <department>
            <name> EE  </name>
        </department>
</person>
<person>
        <name>
            <first> Sang </first>
            <last> Song </last>
        </name>
        <student rank="ugrad" />
        <department>
            <name> ME  </name>
        </department>
</person>
</people>
```

people.xsl

```
<xsl:template match = "/">
   <html xmlns = "http://www.w3.org/1999/xhtml">
      <body>
      Processing people.xml
      <xsl:apply-templates select="people/person" />
      </body>
   </html>
```

… continued on the next page

people.xsl

```
<xsl:template match="name">
        <br/>
        <xsl:value-of select="." />
 </xsl:template>
 <xsl:template match="faculty">
        <br/>
        <xsl:value-of select="./@rank" />
        <xsl:value-of select="../name/last" />
        <xsl:value-of select="." />
</xsl:template>
<xsl:template match="person/name">
        <br/>
        <xsl:value-of select="first" />
</xsl:template>
```

Output produced using people.xsl

```
Processing people.xml
Jack
Prof. Jumping
CS
Mickey
Prof. Mouse Assistant Professor
CS
Larry
CS
Jane
EE
Mocha
EE
Sang
ME
```

people1.xsl

```
<xsl:template match = "/">
   <html xmlns = "http://www.w3.org/1999/xhtml">
      <body>
      Processing people.xml
      <xsl:apply-templates select="people/person" />
      </body>
   </html>
<xsl:template match="name">
        <br/>
        <xsl:value-of select="." />
 </xsl:template>
 <xsl:template match="faculty">
        <br/>
        <xsl:value-of select="./@rank" />
        <xsl:value-of select="../name/last" />
        <xsl:value-of select="." />
 </xsl:template>
```

What will be the output produced by this XSL style file?

Output produced using people.xsl

```
Processing people.xml
 Jack Jumping
 Prof. Jumping
 CS
 Mickey Mouse
 Prof. Mouse Assistant Professor
 CS
 Larry Limping
 CS
 Jane Doe
 EE
 Mocha Coffee
 EE
 Sang Song
 ME
```

Example: Figure 15.21
Page 543-545 Deitel Book

In this example, an XMl document describes a book.

- Chapters are listed but in some random order.
- Similarly, appendices appear in some random order.
- For each chapter, page-count is given in the XML document.

We want to list

- All chapters in their sequential order.
- All appendices in the alphabetical order.
- Total page count for the book needs to be calculated and printed.

Example: Figure 15.20
Page 543 Deitel Book

```
<book isbn = "999-99999-9-X">
  <title>Deitel&apos;s XML Primer</title>
  <author>
    <firstName>Jane</firstName>
    <lastName>Blue</lastName>
  </author>
  <chapters>
    <frontMatter>
      <preface pages = "2" />
      <contents pages = "5" />
      <illustrations pages = "4" />
    </frontMatter>
```

Example: Page 543 Deitel Book

```
    <chapter number = "3"  pages = "44">Advanced XML</chapter>
    <chapter number = "2"  pages = "35">Intermediate XML</chapter>
    <appendix number = "B" pages = "26">Parsers and Tools</appendix>
    <appendix number = "A" pages = "7">Entities</appendix>
    <chapter number = "1" pages = "28">XML Fundamentals</chapter>
  </chapters>
  <media type = "CD" />
</book>
```

- Problem: Chapters and appendix are not appearing in the sequential order.
    - We want to display them in the proper sorted order.
    - We also want to show the total number of pages.
- This will be done in the XSL stylesheet code.
- Click here to see this example.

XSL File (Figure 15.21)

```
<?xml version = "1.0"?>
<xsl:stylesheet version = "1.0"
  xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
  <!-- write XML declaration and DOCTYPE DTD information -->
  <xsl:output method = "html" omit-xml-declaration = "no"
    doctype-system = "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   strict.dtd"
    doctype-public = "-//W3C//DTD XHTML 1.0 Strict//EN"/>
```

…continued on next page

```
<!-- match book -->
  <xsl:template match = "book">
    <head>
      <title>ISBN <xsl:value-of select = "@isbn"/> -  <xsl:value-of select =
    "title"/></title>
    </head>
 <body>
      <h1 style = "color: blue"><xsl:value-of select = "title"/></h1>
      <h2 style = "color: blue">by
        <xsl:value-of select = "author/lastName"/>,
        <xsl:value-of select = "author/firstName"/></h2>
      <table style = "border-style: groove; background-color: wheat">
        <xsl:for-each select = "chapters/frontMatter/*“ >
          <tr>
            <td style = "text-align: right">   <xsl:value-of  select = "name()” /></td>
            <td> ( <xsl:value-of    select = "@pages"/> pages )   </td>
          </tr>
        </xsl:for-each>
```

```
<xsl:for-each    select = "chapters/chapter">
          <xsl:sort    select = "@number" data-type = "number"
              order = "ascending"/>
          <tr>
            <td style = "text-align: right">
              Chapter <xsl:value-of    select = "@number"/>
            </td>

            <td>
              <xsl:value-of    select = "text()"/>
              ( <xsl:value-of    select = "@pages"/> pages )
            </td>
          </tr>
</xsl:for-each>
```

```
<xsl:for-each select = "chapters/appendix">
    <xsl:sort select = "@number" data-type = "text" order = "ascending"/>
     <tr>
     <td style = "text-align: right">  Appendix
             <xsl:value-of select = "@number"/>  </td>
      <td> <xsl:value-of select = "text()"/>
              ( <xsl:value-of select = "@pages"/> pages )
      </td>  </tr>
</xsl:for-each>
</table>
 <br />
<p style = "color: blue">Pages:
       <xsl:variable     name = "pagecount“  select = "sum(chapters/*/@pages)"/>
       <xsl:value-of     select = "$pagecount"/>
<br />Media Type: <xsl:value-of select = "media/@type"/>    </p>
```

Extending the example further

Suppose that we have a list of books in the following form:

```
<booklist>
    <book>  …  </book>
    <book>  …  </book>
</booklist>
```

We now want to print information for each book.

This is example booklist.xml and booklist.xsl

Click here to see the example

XSL Code

```
<xsl:template match = "/booklist">
     <head>
        <title>ISBN <xsl:value-of select = "@isbn"/> -
           <xsl:value-of select = "title"/></title>
     </head>
     <body>
     <xsl:for-each select ="book">
      …… Same code as in the pervious example
      </xsl:for-each>
<xsl:template>
```