XML APIs

Web Data Management and Distribution

Serge Abiteboul Philippe Rigaux
Marie-Christine Rousset Pierre Senellart

http://gemo.futurs.inria.fr/wdmd

January 4, 2010

Application Programming Interfaces (APIs)

DOM, the *Document Object Model*.

- It provides a hierarchical representation, where each node is an object instance of a DOM class.
- Normalized by the W3C (see http://www.w3.org/DOM/).
- DOM parsers exist in all object-oriented language: Java and C++ (the *Xerces* parser, from Apache), JavaScript (Ajax), PHP, Python, etc.
- Not very efficient, and space consumming.

SAX, the *Simple API for XML*.

- Operates on the serialized representation;
- Associates triggers to each syntactic feature (e.g., a tag);
- Efficent (one scan of the serialized representation; not always appropriate).

XML:DB, the XML Database API.

- Provides a common interface to native or XML-enabled databases (i.e., meant as the "JDBC for XML" API);
- Promoted by the XML:DB initiative (see http://xmldb-org.sourceforge.net/xapi/);

Content of this presentation

A bird's eye view of the principles of these APIs, along with a few examples.

Outline

1 Introduction

2 SAX

3 DOM

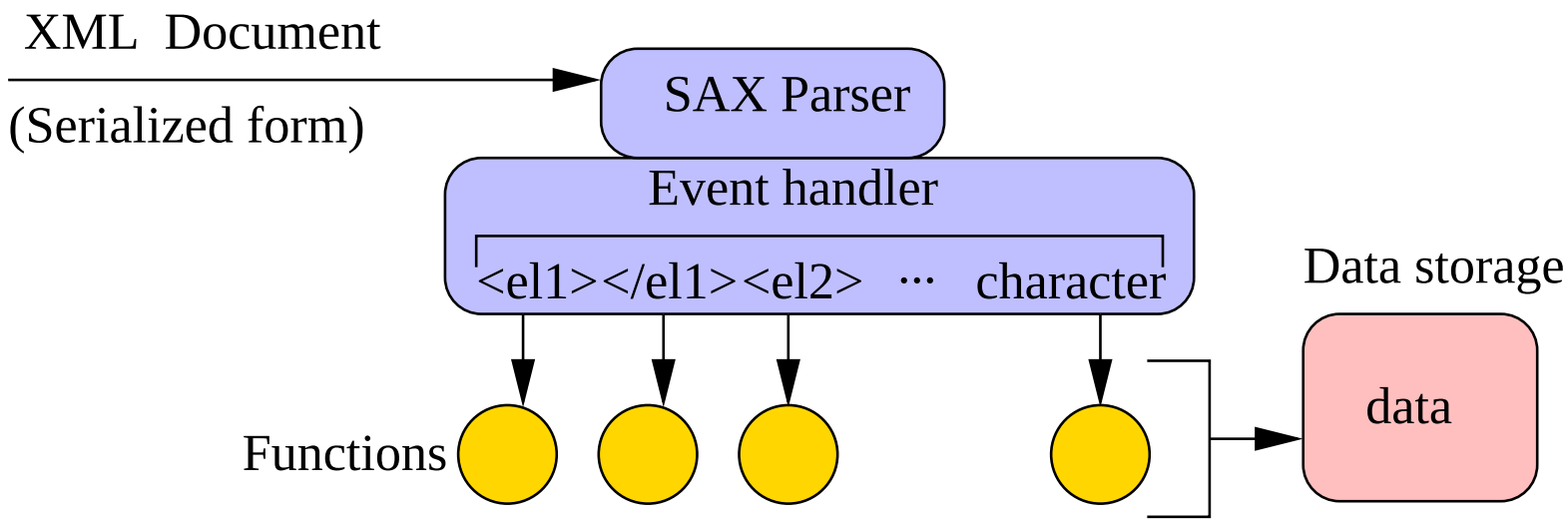
4 XML:DB

SAX: main principles

SAX is the API of choice for processing XML document in *serialized form* (including *XML streams*).

The XML input is read once, and the parser triggers *handlers* when *events* are met.

An *event* is simply a syntactic feature of the document: an opening or a closing tag, a line in a character string, an entity, etc.



SAX example: the Handler

Programming with SAX = writing a *handler* (subclass of the abstract class `ContentHandler`) which defines all the functions that must be triggered.

```
import org.xml.sax.*;
import org.xml.sax.helpers.LocatorImpl;

public class SaxHandler implements ContentHandler
{
    private Locator locator;

    /** Constructor */
   public SaxHandler() {
      super();
      // Set the default locator
      locator = new LocatorImpl();
   }
```

Note: the locator can be used to know the location of the parser when an event is processed.

SAX example: handler functions

Writing a handler = defining methods *startDocument*, *startElement*, *endElement*, etc.

```
/** Opening tag handler */
public void startElement(String nameSpaceURI,
                         String localName,
                         String rawName,
                         Attributes attributes)
throws SAXException {
  System.out.println("Opening tag: " + localName);

  // Show the attributes, if any
  if (attributes.getLength() > 0) {
    System.out.println("  Attributes:  ");
    for (int i = 0; i < attributes.getLength(); i++) {
      System.out.println(attributes.getLocalName(i)
        + " = " + attributes.getValue(i));
    }
  }
}
```

SAX example: handler functions (cont.)

```
/** Closing tag handler */
public void  endElement(String nameSpaceURI,
                        String localName,
                        String rawName) throws SAXException {
    System.out.print("Closing tag : " + localName);
    System.out.println();
  }

  /** Character data handling */
 public void characters(char[] ch,
                        int start, int end)
      throws SAXException {
          System.out.println("#PCDATA: "
            + new String(ch, start, end));
}
```

Calling the SAX handler

```
public class SaxExample {
  /** Constructor */
  public SaxExample(String uri) {
    XMLReader saxReader =
      XMLReaderFactory.createXMLReader(
          "org.apache.xerces.parsers.SAXParser");
    saxReader.setContentHandler(new SaxHandler());
    saxReader.parse(uri);
  }

  public static void main(String[] args) {
    try {
     SaxExample parser = new SaxExample(args[0]);
    } catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

Outline

1 Introduction

2 SAX

3 DOM

4 XML:DB

The DOM approach

According to the DOM, everything in an XML document is a node.

In object-oriented terms: everything is an object, instance of class **Node** or instance of a subclass of **Node**.

1. The entire document is a **Document** node
2. Every XML tag is an **Element** node
3. The texts contained in the XML elements are **Text** nodes
4. Every XML attribute is an **Attribute** node
5. Comments are **Comment** nodes

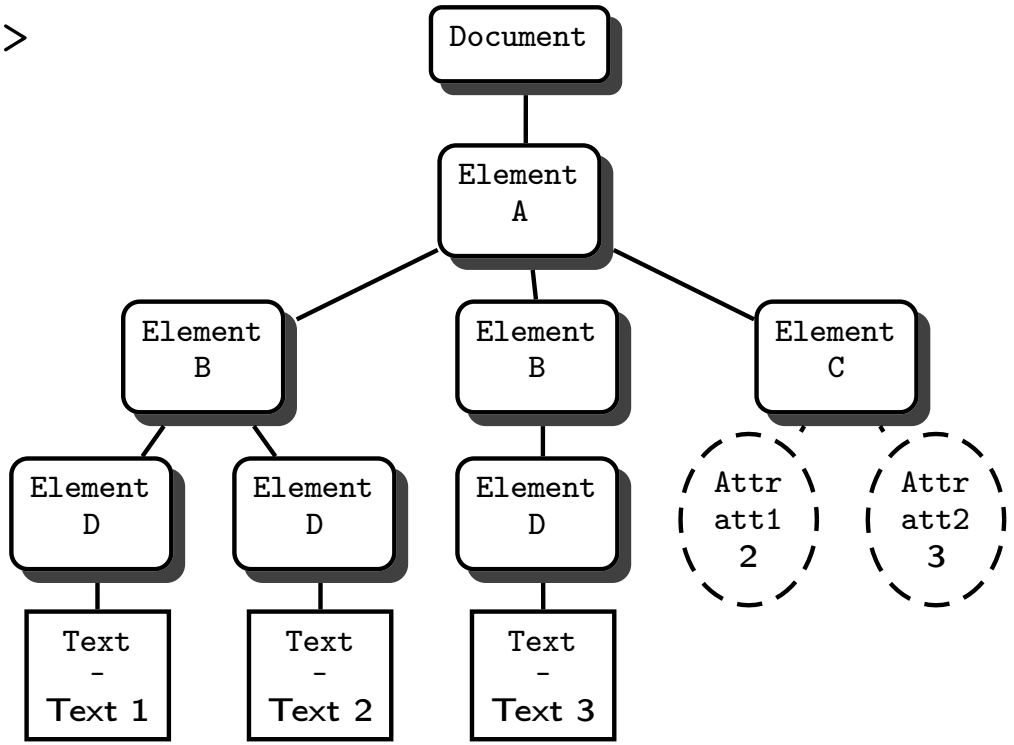
Plus, many other classes, not used for the tree representation.

Remark

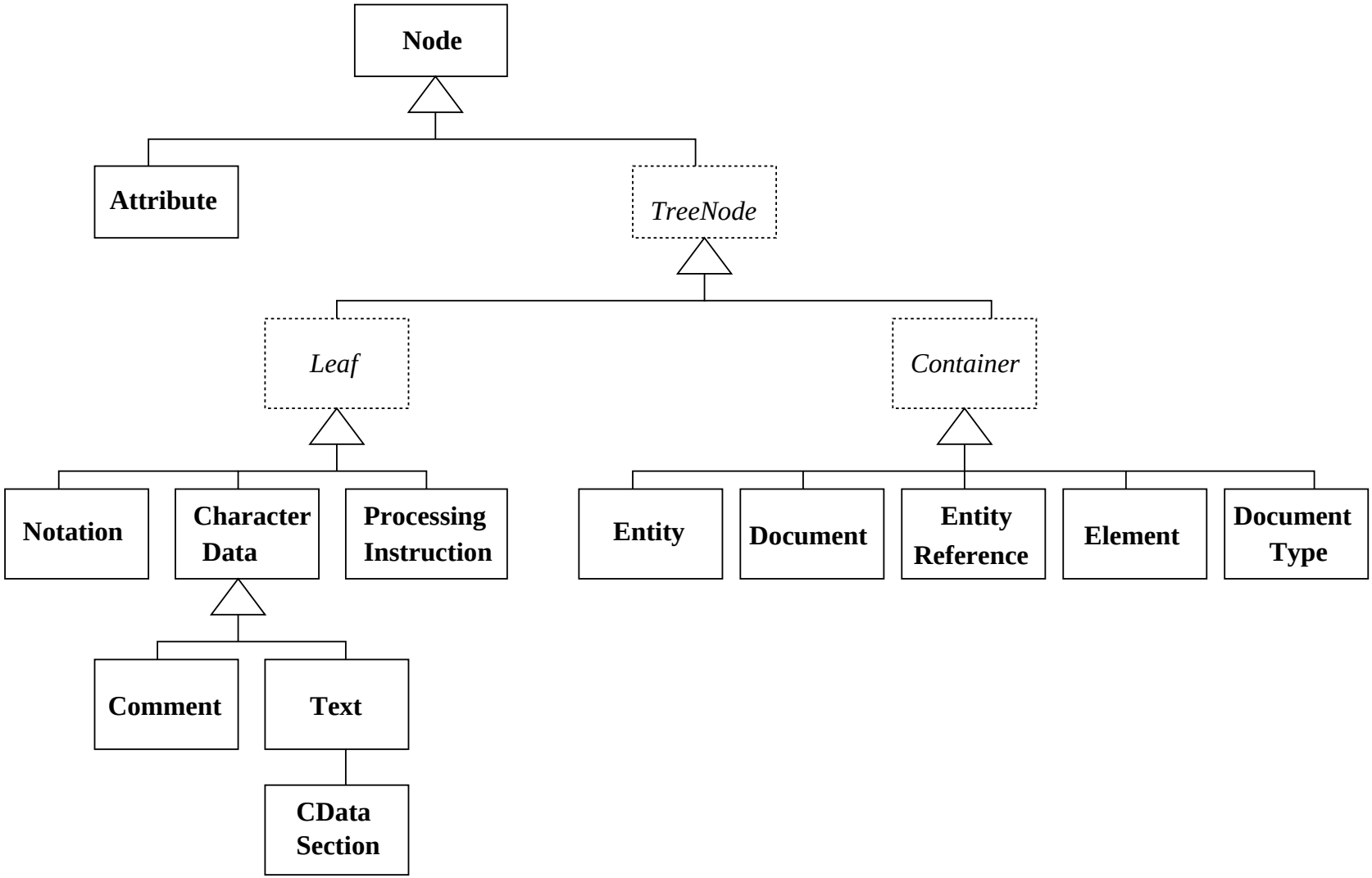
Remember: an **Element** node does not contain the text.

From serialized representation to DOM tree (reminder!)

```
<?xml version="1.0"
    encoding="UTF-8"?>
<A>
  <B>
    <D>Text 1</D>
    <D>Text 2</D>
  </B>
  <B>
    <D>Text 3</D>
  </B>
  <C att1="2"
     att2="3"/>
</A>
```



The DOM hierarchy (excerpt)



The **Node** super-class

DOM is an attempt to provide an object-oriented model of XML document.

Node is the super-class. It should gather all the properties common to all nodes. **But** some properties are properly inherited in a child class, and remain undefined in another.

Example: the name is inherited by **Element** nodes, but is undefined for **Text** node.

Actually there is no obvious OO hierarchy that cleanly models XML trees (i.e, from very abstract to very specialized nodes).

A pragmatic approach

The **Node** provides all the properties of all the node types. Thus one can:

- Adopt the OO paradigm and map as accurately as possible each node to the specialized type;
- or see everything as a **Node**, and follow a more procedural approach.

Properties of the **Node** class

Property	Type	Property	Type
`nodeType`	**short**	`nodeName`	**String**
`nodeValue`	**String**	`parentNode`	**Node**
`firstChild`	**Node**	`lastChild`	**Node**
`childNodes`	**NodeList**	`previousSibling`	**Node**
`nextSibling`	**Node**	`attributes`	**NamedNodeMap**

Methods of the **Node** class

Some important methods of **Node**. Note: the "current node" refers to the object that processes the method.

- *insertBefore* (**Node** *new*, **Node** *child*)
  Inserts the node *new* as a new child of the current node, just before *child*.
- *replaceChild* (**Node** *new*, **Node** *old*).
  Replace the *child* node by *new*.
- *removeChild* (**Node** *child*)
  Remove a child node;
- *appendChild* (**Node** *child*)
  Ad a child node in last position (i.e., after the last of the current children).
- **boolean** *hasChildNodes*().
  True, if the current node has children.

Methods of the **Document** class

A **Document** object is always the first node created for a new XML tree.

Therefore it plays the role of a factory for creating new nodes that must be inserted in the tree.

Methods of **Document**:

- *createElement()*: creates and returns an **Element** node;
- *createTextNode()*: creates and returns an **Text** node;
- *createCommentNode()*: creates and returns an **Comment** node;
- etc.

A first example: the *preorder* DOM program

preorder is a simple DOM program that

1. instantiate a DOM parser (Xerces);
2. Traverse a DOM tree in preorder;
3. Add to each **Text** node its position in the preorder traversal;
4. Serializes the output.

Remark

The program is available on the web site. Tested with the Xerces parser. Should work with any other parser!

First step: instantiate the parser

```
// Import Java classes
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

class DomPreorder
{
 public static void main (String args [])
 {
   try
   {
      // Instantiate the DOM parser
      DocumentBuilderFactory factory =
         DocumentBuilderFactory.newInstance();
      DocumentBuilder builder =
        factory.newDocumentBuilder();

      (: see next slide :)
   }
```

Second step: call the preorder recursive method

Note two important initial initial expressions: one gets the **Document** node as result of the *parse* method, and the root **Element** node as result of the *getDocumentElement* method.

```
    // Analyse the document
    File fdom = new File (args[0]);
    Document dom = builder.parse(fdom);
    Node rootElement = dom.getDocumentElement();

    // Start the pre-order scan.
    // The first node number is 1.
    exploreInPreorder (rootElement, 1);

  // Serialize the result
  DomSerializer serializer = new DomSerializer (dom);
  serializer.output ("Output.xml");
```

The *exploreInPreorder* method

```
private static int
    exploreInPreorder (Node node, int number)
{
  String str = new String();
  number++;

  // If Text node: put the number in the node's value.
  if (node.getNodeType() == Node.TEXT_NODE) {
    str = "(" + number + ") " + node.getNodeValue();
    node.setNodeValue (str);
  }

  // Recursive call
  (: see next slide :)

  return number;
}
```

The recursive call

```
private static int
    exploreInPreorder (Node node, int number)
{
  (: see prevous slide :)

  // Recursive call
  if (node.hasChildNodes()) {
    // Get the children of the current node
    NodeList children = node.getChildNodes();

    // Pre-order traversal for each node in the list
    for (int i=0; i < children.getLength(); i++)
      number =
        exploreInPreorder (children.item(i), number);
    }
  return number;
}
```

Outline

1 Introduction

2 SAX

3 DOM

4 XML:DB

Main components of XML:DB

The basic components employed by the XML:DB API are drivers, collections, and resources.

Drivers are implementations of the database interface that encapsulate the database access logic for specific XML database products.
They are provided by the product vendor and must be registered with the database manager.

Collections are hierarchical containers for resources and further sub-collections.

Resources represent an XML document or a document fragment, selected by a query.

Remark

Our examples have been tested with eXist. See the site for the code, and further instructions.

First XML:DB example: retrieving a document

The database driver class for *eXist* is `org.exist.xmldb.DatabaseImpl`

The URI gives the address of the *eXist* instance, and the access protocol.

```
public class ExistAccess {
 String DRIVER = "org.exist.xmldb.DatabaseImpl";
 String URI = "xmldb:exist://localhost:8080/exist/xmlrpc
 String collectionPath = "/db/movies/";
 String resourceName = "Heat.xml";

  (: see next slide :)
```

Remark

See the actual code for `import` instructions.

First XML:DB example (continued)

```
public static void main(String[] args) throws Exception
{
  // initialize database driver
  Class cl = Class.forName(DRIVER);
  Database database = (Database) cl.newInstance();
  DatabaseManager.registerDatabase(database);

  // get the collection
  Collection col =
  DatabaseManager.getCollection(URI + collectionPath);

  //get the content of a document
  System.out.println("Get   " + resourceName);
  XMLResource res = col.getResource(resourceName);
  System.out.println(res.getContent());
}
```

Secondt XML:DB example: execute a query

```
(: Declarations and initializations as before :)

//query a document
String xQuery=
    "for $x in doc('movies.xml')//title return $x";

// Instantiate a XQuery service
XQueryService service = col.getService("XQueryService"

// Execute the query, print the result
ResourceSet result = service.query(xQuery);
ResourceIterator i = result.getIterator();
while(i.hasMoreResources()) {
  Resource r = i.nextResource();
  System.out.println((String)r.getContent());
}
```