Data-centric XML

Parser Interfaces: SAX

XML Programming Models

- Treat XML as text
  - useful for interactive creation of documents (text editors)
  - also useful for programmatic generation of documents
    - need guarantee for well-formedness somehow
  - very difficult for XML input
    - in certain application contexts, additional constraints on XML may simplify processing, e.g. restriction to us-ascii, avoiding CDATA sections, etc.

XML Programming Models (2)

- Treat XML as events
  - use separate XML parser (e.g. as a library)
  - parser invokes callback functions for parsing “events”:
    - start of an element, including attributes
    - end of an element
    - character data content
    - …
  - applications need to maintain state
  - advantage: parser does not need to maintain state, can thus process large documents quickly

XML Programming Models (3)

- Treat XML documents as trees
  - Allows random access to all parts of the document
  - Requires potentially large in-memory representation
  - Different notions of how to represent XML in a tree:
    - XML Infoset
    - DOM (Document Object Model)
    - XPath
    - XSLT: Post-Schema Validation Infoset (PSVI)
  - Tree structure typically accessed through APIs

XML Programming Models (4)

- Transformation
  - Application assumption: Output is again a structure document (e.g. XML)
  - Processing is defined by means of rules associating input and output
  - XSLT: Pattern-matching over input document, templates for output document
  - Used either stand-alone, or embedded in some larger application

XML Programming Models (5)

- Abstracting XML away
  - Use some tool/library to produce application-oriented data representation
  - Use XML “under the hood”:
    - Web Services
    - XMI: XML Model Interchange

Common Problems

- What you get is not what you saw
  - parsers abstract from lexical representation
    - order of attributes
    - expansion of CDATA sections and references
    - retrieval of default values for attributes
    - …
- Omission of certain constructs (e.g. comments)
  - Applications should not use comments to store essential information
- Application needs out-of-band information
  - processing instructions
  - notations
  - unparsed entities

SAX: Simple API for XML

- Event-oriented programming interface
- Originally specified by Tim Bray, Peter Murray-Rust, and David Megginson for Java
  - Java package org.xml.sax
- Current version: SAX 2
  - added support for namespaces
- Spec available at http://www.saxproject.org/
- Implementations available for many languages
  - Java, Python, Perl, Eiffel, …
- Many implementations available
  - Apache Xerces (C++ & Java), MSXML, Oracle XML Parser, PyXML, …

SAX 2 Overview

- Interfaces implemented by the processor (reader)
  - XMLReader: primary interface
  - Attributes: access to attributes in start-tag callback
  - Locator (optional): access to source URI+line position
  - XMLReaderFactory: creation of new readers
- Interfaces implement by the application
  - ContentHandler: primary interface, for tags and character data
  - DTDHandler: receives basic DTD events (notations, unparsed entities)
  - EntityResolver: Must provide input stream for external entities
  - ErrorHandler: invoked upon errors in the document

SAX 2 Overview (2)

- Standard classes: objects encapsulating processing state
  - InputSource: carry public and system identifiers
  - SaxException: base class for SAX errors
    - SAXParseException, SAXNotSupportedException, SAXNotRecognizedException
  - DefaultHandler: Base class implementing all handlers
- Standard Extensions: available only in some implementations
  - Attributes2: Access to DTD information about attributes
  - DeclHandler: Callback for DTD declarations (attributes, elements, entities)
  - EntityResolver2: advanced resolution of entities and external DTD subset
  - LexicalHandler: callback for various lexical information (comments, CDATA sections)
  - Locator2: additional source information (XML version, encoding)

SAX Overview (3)

- Features and Properties: Configuration of parser
  - whether or not to perform validation
  - whether or not to provide namespace URIs
  - whether or not to use various extension interfaces

Obtaining a Reader

```
try{
  // optionally pass class name, e.g. “org.apache.xerces.parsers.SAXParser”
  XMLReader parser = XMLReaderFactory.createXMLReader();
  MyHandler handler = new MyHandler();
  parser.setContentHandler(handler);                    // install a custom handler
  parser.parse(“http://www.slashdot.org/slashdot.xml”); // parse the document
  handler.saveResults();                                // retrieve results from handler
} catch(SAXParseException e){
  // syntax error in document
} catch(SAXException e){
  // no parsers found
} catch(IOException e){
  // Error accessing the resource
}
```

Content Handlers

```
package org.xml.sax;
public interface ContentHandler {
  public abstract void setDocumentLocator (Locator locator);
  public abstract void startDocument () throws SAXException;
  public abstract void endDocument () throws SAXException;

  public void startPrefixMapping (String prefix, String uri) throws
  SAXException;
  public void endPrefixMapping (String prefix) throws SAXException;
```

Content Handlers (2)

```
  public abstract void startElement (String namespaceURI, String localName,
                                     String qName, Attributes atts) throws SAXException;
  public abstract void endElement (String namespaceURI, String localName,
                                   String qName) throws SAXException;
  public abstract void characters (char ch[], int start, int length)
                                 throws SAXException;
  public abstract void ignorableWhitespace (char ch[], int start, int length)
        throws SAXException;
  public void processingInstruction (String target, String data)
        throws SAXException;
  public void skippedEntity (String name) throws SAXException;
}
```

Content Handlers (3)

- setDocumentLocator: optionally called by parser before calling anything else
- startDocument/endDocument: called exactly once
- startPrefixMapping/endPrefixMapping: provide current prefixes
  - for expansion of prefixed attribute values(!)
- startElement/endElement: provide element name and namespace information
  - namespace URI/localName optional depending on namespaces property
  - attributes only includes explicit attributes (specified or defaulted)
  - xmlns attributes provided only if the namespace-prefixes feature is true

Content Handlers (4)

- characters: provide PCDATA as a Unicode string
  - reader may split characters over several calls (!), e.g. for line breaks, CDATA sections, references
  - all data in a single call must come from the same external entity, so that locator information is useful
- ignorableWhitespace: reports white space in element content
  - usage is optional, must be used by validating parsers
- skippedEntity: report entity reference that is not expanded
  - may be called by non-validating parsers for any external entity
  - may also be called if external-general-entities/external-parameter-entities is false.

XMLReader: Configuration

```
package org.xml.sax;
public interface XMLReader
{
  public boolean getFeature (String name)
        throws SAXNotRecognizedException, SAXNotSupportedException;
  public void setFeature (String name, boolean value)
        throws SAXNotRecognizedException, SAXNotSupportedException;
  public Object getProperty (String name)
        throws SAXNotRecognizedException, SAXNotSupportedException;
  public void setProperty (String name, Object value)
        throws SAXNotRecognizedException, SAXNotSupportedException;
```

XMLReader: Event Handlers

```
public void setEntityResolver (EntityResolver resolver);
public EntityResolver getEntityResolver ();

public void setDTDHandler (DTDHandler handler);
public DTDHandler getDTDHandler ();

public void setContentHandler (ContentHandler handler);
public ContentHandler getContentHandler ();

public void setErrorHandler (ErrorHandler handler);
public ErrorHandler getErrorHandler ();
```

XMLReader: Parsing

```
 public void parse (InputSource input)
        throws IOException, SAXException;

 public void parse (String systemId)
        throws IOException, SAXException;
}
```

InputSources

```
package org.xml.sax;
public class InputSource {
  public InputSource();
  public InputSource (String systemId);
  public InputSource (java.ioInputStream byteStream);
  public InputSource (java.io.Reader characterStream);
// getters omitted
  public void setPublicId (String publicId);
  public void setSystemId (String systemId);
  public void setByteStream (InputStream byteStream);
  public void setEncoding (String encoding);
  public void setCharacterStream (Reader characterStream);
};
```

Locators

```
package org.xml.sax;
public interface Locator {
  public abstract String getPublicId ();
  public abstract String getSystemId ();
  public abstract int getLineNumber ();
  public abstract int getColumnNumber ();
}
```

- positions are only approximations; they cannot be used for editing

SAX Features

- Feature names are URLs
  - start with http://xml.org/sax/features/
  - external-general-entities
  - external-parameter-entities
  - is-standalone (read-only)
  - namespaces (default: true)
  - namespace-prefixes (default:false)
  - resolve-dtd-uris (default:true)
  - string-interning
  - validation
  - xmlns-uris (false)
  - …

Filters

- Receive all events and delegate them to a next handler in a processing chain
- allow modular development of XML applications
- Base class: org.xml.sax.helpers.XMLFilterImpl
  - implements both reader and handler interfaces
- Install “parent” reader with .setParent
- Invoke “outermost” .parse()

Alternative Implementation Language

- MSXML4, using C++

```
struct MyContent : public SAXContentHandlerImpl  {
 MyContent();
        virtual ~MyContent();

 virtual HRESULT STDMETHODCALLTYPE startElement(
        wchar_t __RPC_FAR *pwchNamespaceUri,     int
cchNamespaceUri,
        wchar_t __RPC_FAR *pwchLocalName,         int cchLocalName,
        wchar_t __RPC_FAR *pwchQName,             int cchQName,
        ISAXAttributes __RPC_FAR *pAttributes);
```

```
    virtual HRESULT STDMETHODCALLTYPE endElement(
      wchar_t __RPC_FAR *pwchNamespaceUri,   int
 cchNamespaceUri,
      wchar_t __RPC_FAR *pwchLocalName,       int cchLocalName,
      wchar_t __RPC_FAR *pwchQName,           int cchQName);

    virtual HRESULT STDMETHODCALLTYPE startDocument();
};
```

```
HRESULT STDMETHODCALLTYPE MyContent::startElement(...)
{
 int l; wchar_t * ln, * vl; int lnl, vll;
 prt(L"<%s",pwchLocalName,cchLocalName);
        pAttributes->getLength(&l);
        for ( int i=0; i<l; i++ ) {
                pAttributes->getLocalName(i,&ln,&lnl);
        prt(L" %s=", ln, lnl);
        pAttributes->getValue(i,&vl,&vll);
                prt(L"\"%s\"", vl, vll);
 }
        printf(">");
 return S_OK;
}
```