Interacción entre Aplicaciones: XML

En esta práctica se va a conocer y experimentar con la manipulación de documentos XML y las herramientas que hay disponibles.

Muchas aplicaciones utilizan XML para varios propósitos:

- Presentación: la separación entre contenido, los datos, y presentación, el formato de los datos. Por ejemplo los datos de una reserva de avión se han de presentar en html para un cliente web, en WML o XHTML para un dispositivo móvil, o en PDF si se va a imprimir. Por ejemplo, en la construcción de clientes y servidores Web que usen XML (como cocoon, enhydra, caucho, bluestone, saxon) que transforman en función del cliente xml → xml+xsl | xhtml | wml | pdf.
- Comunicación: entre aplicaciones sin que el formato esté ligado a ninguna presentación. En muchas ocasiones comunidades de organizaciones o empresas de un cierto sector definen un vocabulario XML común para la interacción entre todos ellos de documentos como pedidos, facturas, etc. (e-business)
- Interacción: intercambio de documentos XML en forma de llamada a función remota como XML-RPC o SOAP en que dos componentes intercambian documentos XML que invocan una operación remota.
- Configuración: para guardar datos de configuración de las aplicaciones, como hace Apache, Tomcat o Enterprise Java Beans (EJB), o para guardar los datos de cualquier aplicación.

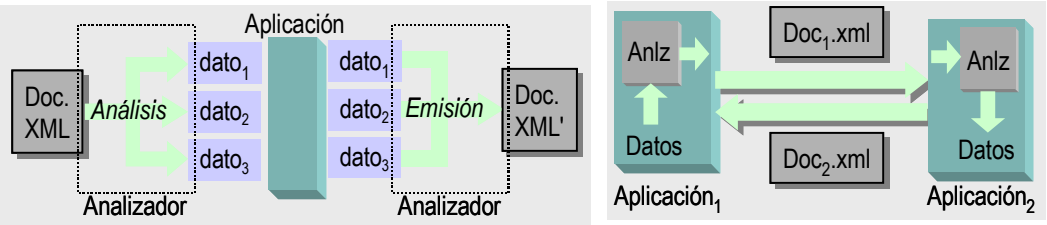

Fig 1.- Usos de XML: 1.a) configuración, transformación; 1.b) intercambio entre aplicaciones.

La buena noticia, si se trabaja con información representada en xml, es que abundan las herramientas para analizar texto xml y acceder, tratar, transformar los datos que contiene. Para ello se utilizan componentes que suelen ofrecen uno o los 2 API estandarizados: SAX y DOM. Si se utiliza uno de estos API, el código que utiliza un analizador no cambia si se sustituue el componente que procesa documentos XML.

Un analizador XML ofrece los servicios de serializar y deserializar documentos XML, entre la forma serie (textual) y la forma no serie seleccionada por el programador: una estructura de datos o un conjunto de llamadas a métodos, uno por cada componente del documento. Estos mecanismos de serialización y deserialización se utilizan también para construir RPC con datos en XML sobre transporte HTTP.

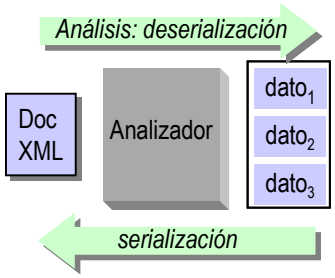

Fig 2.- procesos que realiza un analizador

Se va a utilizar el analizador XercesJ_1_2, disponible en C, perl y Java y hecho público por IBM bajo licencia Apache. Se puede encontrar en `http://xml.apache.org` (y para facilitar el acceso, una copia en el web del laboratorio de aad)

Los API más habituales son:

- SAX (Simple API for XML) funciona por eventos y métodos asociados. A medida que el analizador va leyendo el documento xml y encuentra (los eventos) los componentes del documento

(elementos, atributos, valores, etc) o detecta errores, va invocando a las funciones que ha asociado el programador. Más información de SAX en `http://www.megginson.com/SAX`

- DOM (Document Object Model) Mientras que SAX ofrece acceso a cada trozo del documento xml tal como se va leyendo, DOM proporciona una representación de un documento XML en forma de árbol. El árbol se puede recorrer y transformar. El principal inconveniente es también el árbol: 1) sólo se accede a los datos una vez se han leído todos, y 2) el árbol es un objeto cargado en memoria; esto es problemático para documentos grandes y complejos. Más información de DOM en `http://www.w3.org/DOM`

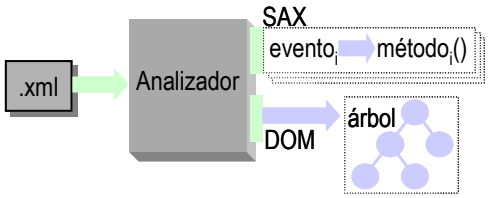

Fig 3.- Analizador

- JDOM: un API para leer, escribir, crear y manipular XML cómodamente desde Java. Al ser un API específico para Java es mucho más intuitivo y sencillo que los anteriores, pero no está pensado para otros lenguajes. Más información de JDOM en `http://www.jdom.org`

Los analizadores se pueden adaptar y seleccionar diferentes opciones, entre ellas si ha de comprobarse o no la validez de un documento xml respecto a su definición (el DTD). Esta validación añade una comprobación y por tanto una garantía más, pero también consume más recursos y ralentiza el procesamiento del documento xml. En cada situación habrá que valorar qué opciones son adecuadas.

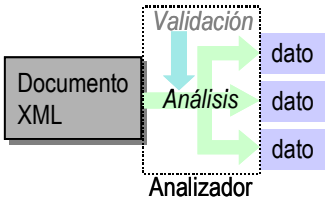

Fig. 4.- Los analizadores pueden también validar el documento xml.

Las aplicaciones que se basan en la transformación de documentos XML, también pueden recurrir a un procesador para transformar documentos XML según las reglas e instrucciones de XSLT, de forma que la transformación de un documento XML se rige por una hoja de estilo XSL (que también es un documento XML), en lugar de escribir un programa que lo haga. Esto queda fuera del ámbito de esta práctica.

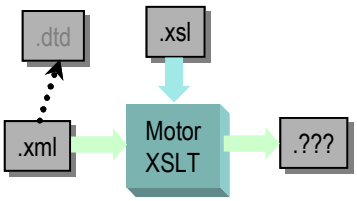

Fig. 5: XSLT.

Análisis de XML con SAX y DOM

Un programa sencillo va a contar el número de elementos y atributos de un documento xml.

Con SAX:

1) Instanciar un "lector" que implemente el interface `org.xml.sax.XML-Reader`, que en Xerces es la clase `org.apache.xerces.parsers.SAXParser`; invocar el método `parse` para que analice el documento pasado como argumento del programa.

```
import org.xml.sax.XMLReader;
import org.apache.xerces.parsers.SAXParser;
public class SAXParserMin {
  public static void main(String [] args) {
    if (args.length != 1) {
      System.out.println("Uso: java SAXParserMin url");
```

```
      System.exit(0);
    }
    XMLReader parser = new SAXParser();
    parser.parse(url);
  }
}
```

Si el programa funciona, quiere decir que java y xerces están bien instalados. Sin embargo el programa no hace nada pues no hemos registrado interés en ningún evento durante el análisis.

Dado que el documento a analizar puede fallar durante la transferencia o ser incorrecto, es conveniente capturar las excepciones `java.io.IOException` y `org.xml.sax.SAXException` al invocar al método `parse`.

Gestores de Contenido

Hay cuatro interfaces o familias de funciones que se pueden asociar a eventos: `org.xml.sax.ContentHandler`: eventos sobre datos (el principal y el más extenso), `org.xml.sax.ErrorHandler`: eventos sobre errores, `org.xml.sax.DTDHandler`: tratamiento de DTDs y por último `org.xml.sax.EntityResolver`: entidades externas.

Se pueden definir clases con los métodos necesarios para tratar cada evento que nos interese. Para detectar:

- datos: implementar el interface `org.xml.sax.ContentHandler`, y registrarse en el analizador con el método `setContentHandler()`.
- errores: implementar el interface `org.xml.sax.ErrorHandler`, y registrarse en el analizador con el método `setErrorHandler()`.

Veamos a continuación un ejemplo de analizador SAX que utiliza xerces y que cuenta los elementos, atributos y blancos que aparecen en un documento xml:

```
package sax;

import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;

public class SAX2Count extends DefaultHandler {

  private static final String  DEFAULT_PARSER_NAME =
"org.apache.xerces.parsers.SAXParser";

  private long elements;            /** Elementos */
  private long attributes;          /** Atributos */
  private long characters;          /** Caracteres */
  private long ignorableWhitespace; /** espacios en blanco */

  /** Prints the output from the SAX callbacks. */
  public static void print(String uri) {
   try {
    SAX2Count counter = new SAX2Count();

    XMLReader parser = new org.apache.xerces.parsers.SAXParser();
    parser.setContentHandler(counter);
    parser.setErrorHandler(counter);

    parser.setFeature( "http://xml.org/sax/features/validation", false);
    parser.setFeature( "http://xml.org/sax/features/namespaces", true);
    parser.setFeature( "http://apache.org/xml/features/validation/schema", true);
```

```
    long before = System.currentTimeMillis();
    parser.parse(uri);
    long after = System.currentTimeMillis();
    counter.printResults(uri, after - before);
   } catch (org.xml.sax.SAXParseException spe) {
    spe.printStackTrace(System.err);
   } catch (org.xml.sax.SAXException se) {
    if (se.getException() != null)
      se.getException().printStackTrace(System.err);
    else se.printStackTrace(System.err);
   } catch (Exception e) {
     e.printStackTrace(System.err);
   }
  }

  // DocumentHandler methods

  public void startDocument() {
    elements = 0;
    attributes = 0;
    characters = 0;
    ignorableWhitespace = 0;
  }

  public void startElement(String uri, String local, String raw, Attributes
attrs) {
    elements++;
    if (attrs != null) attributes += attrs.getLength();
  }

  public void characters(char ch[], int start, int length) {
    characters += length;
  }

  public void ignorableWhitespace(char ch[], int start, int length) {
    ignorableWhitespace += length;
  }

  // ErrorHandler methods

  public void warning(SAXParseException ex) {
    System.err.println("[Aviso] "+getLoc(ex)+": "+ ex.getMessage());
  }

  public void error(SAXParseException ex) {
    System.err.println("[Error] "+getLoc(ex)+": "+ex.getMessage());
  }

  public void fatalError(SAXParseException ex) throws SAXException {
    System.err.println("[Error!] "+getLoc(ex)+": "+ex.getMessage());
  }

  private String getLoc(SAXParseException ex) {
    StringBuffer str = new StringBuffer();

    String systemId = ex.getSystemId();
    if (systemId != null) {
      int index = systemId.lastIndexOf('/');
      if (index != -1) systemId = systemId.substring(index + 1);
      str.append(systemId);
    }
//   str += ":" + ex.getLineNumber() + ":" + ex.getColumnNumber();
    str.append(':');
    str.append(ex.getLineNumber());
    str.append(':');
    str.append(ex.getColumnNumber());
```

```
    return str.toString();
  }

  // Public methods

  public void printResults(String uri, long time) {
    // filename.xml: 631 ms (4 elems, 0 attrs, 78 spaces, 0 chars)
    System.out.println (uri+":"+time+"time ms ("
     +elements+" elems, "+attributes+" attrs, "
     +ignorableWhitespace+" spaces, "+characters+" chars)");
    }

  public static void main(String argv[]) {
    print(argv[0]);
  }
}
```

Una vez comprobado que la JVM funciona (ejecutar `javac`), se ha de configurar el CLASSPATH para que encuentre a las librerías de Xerces y JDOM (hacer que `CLASSPATH=~/xerces/xerces.jar;/../jdom.jar` que será diferente en cada entorno), entonces se puede copiar el programa anterior y probar que funciona correctamente. Hay una copia del programa y de Xerces en el web del laboratorio de AAD (http://www.ac.upc.es/docencia/FIB/AAD/lab)

De forma alternativa, utilizando la forma de analizar xml del API DOM (que también implementa Xerces) el código sería así:

```
package dom;

import util.Arguments;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import org.apache.xerces.dom.TextImpl;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class DOMCount {

  private static final String
  private static boolean setValidation    = false; //defaults
  private static boolean setNameSpaces    = true;
  private static boolean setSchemaSupport = true;
  private static boolean setDeferredDOM   = true;

  private long elements; /** Elements. */
  private long attributes; /** Attributes. */
  private long characters; /** Characters. */
  private long ignorableWhitespace; /** Ignorable whitespace. */

  /** Counts the resulting document tree. */
  public static void count(String uri) {
   try {
    DOMParserWrapper parser = new dom.wrappers.DOMParser();
    DOMCount counter = new DOMCount();
    long before = System.currentTimeMillis();
    parser.setFeature( "http://apache.org/xml/features/dom/defer-node-
expansion", setDeferredDOM );
    parser.setFeature( "http://xml.org/sax/features/validation", setValidation);
    parser.setFeature( "http://xml.org/sax/features/namespaces", setNameSpaces);
    parser.setFeature( "http://apache.org/xml/features/validation/schema",setSchemaSupport);

    Document document = parser.parse(uri);
```

```
   counter.traverse(document);
   long after = System.currentTimeMillis();
   counter.printResults(uri, after - before);
  } catch (org.xml.sax.SAXParseException spe) {
  } catch (org.xml.sax.SAXNotRecognizedException ex ){
  } catch (org.xml.sax.SAXNotSupportedException ex ){
  } catch (org.xml.sax.SAXException se) {
   if (se.getException() != null)
            se.getException().printStackTrace(System.err);
   else se.printStackTrace(System.err);
  } catch (Exception e) {
   e.printStackTrace(System.err);
  }
 }

 /** Traverses the specified node, recursively. */
 public void traverse(Node node) {
   if (node == null) return; // is there anything to do?

   switch (node.getNodeType()) {
    case Node.DOCUMENT_NODE: // print document
     elements            = 0;
     attributes          = 0;
     characters          = 0;
     ignorableWhitespace = 0;
     traverse(((Document)node).getDocumentElement());
     break;

    case Node.ELEMENT_NODE: {// print element with attributes
     elements++;
     NamedNodeMap attrs = node.getAttributes();
     if (attrs != null) attributes += attrs.getLength();
     NodeList children = node.getChildNodes();
     if (children != null) {
       int len = children.getLength();
       for (int i = 0; i < len; i++) traverse(children.item(i));
     }
    } break;

    case Node.ENTITY_REFERENCE_NODE: {
     NodeList children = node.getChildNodes();
     if (children != null) {
       int len = children.getLength();
       for (int i = 0; i < len; i++) traverse(children.item(i));
     }
    } break;

    case Node.CDATA_SECTION_NODE: // print text
     characters += node.getNodeValue().length();
     break;

    case Node.TEXT_NODE:
     if (node instanceof TextImpl) {
       if (((TextImpl)node).isIgnorableWhitespace())
         ignorableWhitespace += node.getNodeValue().length();
       else characters += node.getNodeValue().length();
     } else characters += node.getNodeValue().length();
     break;
   }
 }

 public void printResults(String uri, long time) {
 // filename.xml: 631 ms (4 elems, 0 attrs, 78 spaces, 0 chars)
   System.out.println (uri+":"+time+"time ms ("
    +elements+" elems, "+attributes+" attrs, "
    +ignorableWhitespace+" spaces, "+characters+" chars)");
 }
```

```
  public static void main(String argv[]) {
    count(argv[0] ); //count uri
  }
}
```

Los API anteriores están disponibles para multitud de lenguajes de programación, pero a cambio de la portabilidad del API, el uso es engorroso desde Java. Se ha diseñado una librería específica para Java (JDOM), pero que a cambio utiliza las capacidades particulares del lenguaje Java, que simplifica notablemente la programación. Puede encontrarse más información en `http://www.jdom.org`

El ejemplo anterior escrito usando el API JDOM sería:

```
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import org.jdom.input.DOMBuilder;
import org.jdom.output.*;
import java.util.*;

public class Count {

    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Usage: java Count URL");
        }

        SAXBuilder saxBuilder = new SAXBuilder();
        DOMBuilder domBuilder = new DOMBuilder();

        System.out.println("File\tElements\tAttributes\tComments\tProcessing
Instructions\tCharacters");

        // start parsing...
        try {
          Document jdomDocument = saxBuilder.build(args[0]);
          DOMOutputter domOutputter = new DOMOutputter();

          /* Test getting DOM Document from JDOM Document
             org.w3c.dom.Document domDocument = domOutputter.output(doc);
           */
          // Test getting DOM Element from JDOM Element
          org.w3c.dom.Element domElement =
             domOutputter.output(jdomDocument.getRootElement());

          // Test getting JDOM Element from DOM Element
          org.jdom.Element jdomElement = domBuilder.build(domElement);
          count(jdomElement);

        } catch (JDOMException e) { // indica doc mal formado u otro error
          System.out.println("Documento XML mal formado o incorrecto.");
          System.out.println(e.getMessage());
        }
    }

    private static int numCharacters             = 0;
    private static int numComments               = 0;
    private static int numElements               = 0;
    private static int numAttributes             = 0;
    private static int numProcessingInstructions = 0;

    public static String count(Document doc) {
        numCharacters = 0;
        numComments = 0;
        numElements = 0;
        numAttributes = 0;
        numProcessingInstructions = 0;

        List children = doc.getMixedContent();
        Iterator iterator = children.iterator();
        while (iterator.hasNext()) {
```

```
            Object o = iterator.next();
            if (o instanceof Element) {
                numElements++;
                count((Element) o);
            }
            else if (o instanceof Comment) numComments++;
            else if (o instanceof ProcessingInstruction)
                        numProcessingInstructions++;
        }

        String result = numElements + "\t" + numAttributes + "\t" +
                        numComments + "\t" + numProcessingInstructions +
                        "\t" + numCharacters;
        return result;
    }

    public static void count(Element element) {

        List attributes = element.getAttributes();
        numAttributes += attributes.size();
        List children = element.getMixedContent();
        Iterator iterator = children.iterator();
        while (iterator.hasNext()) {
            Object o = iterator.next();
            if (o instanceof Element) {
                numElements++;
                count((Element) o);
            }
            else if (o instanceof Comment) numComments++;
            else if (o instanceof ProcessingInstruction)
                        numProcessingInstructions++;
            else if (o instanceof String) {
                String s = (String) o;
                numCharacters += s.length();
            }
        }

        String result = numElements + "\t" + numAttributes + "\t" +
                        numComments + "\t" + numProcessingInstructions +
                        "\t" + numCharacters;
        System.out.println(result);
    }
}
```

Probar los ejemplos anteriores. Para probar JDOM, se ha de descargar la librería `jdom.jar` (del web de lab) y colocarla en el `CLASSPATH`. En el web de laboratorio podréis encontrar páginas de ayuda sobre JDOM.

Tienda de muebles en XML

Ahora se trata de diseñar los intercambios entre compradores, proveedores y la tienda en forma xml. Para ello a partir de un documento xml de partida (en el web de lab), hay que diseñar el DTD que le corresponda.

Para poder probar que el DTD es correcto, se podría hacer un analizador DOM que validara el documento y avisara de los posibles errores. En lugar de programarlo, se puede usar el analizador XML (que incluye el API DOM) incluido a partir de la versión 5.0 de MS Internet Explorer. Se puede crear el fichero `shop.dtd` en el mismo directorio donde esté `shop.xml`, y pedirle a MSIE que abra el fichero local `shop.xml`. El MSIE verificará la sintaxis del DTD, os indicará los errores, y si el DTD es correcto, verificará si el documento xml cumple con la especificación. Así se puede comprobar cómodamente que el DTD está bien escrito y describe bien a `shop.xml`.

`Shop.xml` tal como lo presenta MSIE tras validar `shop.dtd` y verificar que `shop.xml` está bien formado.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE shop (View Source for full doctype...)>
- <list>
  - <op name="Supply">
      <item precio="123456" unidades="60">mesa comedor haya</item>
      <item precio="13456" unidades="100">mesa cocina nogal</item>
      <item precio="13457" unidades="200">silla cocina nogal</item>
    </op>
    <res name="Supply" />
  - <op name="Order" u="juan" pass="nauj">
      <item unidades="1">mesa comedor haya</item>
      <item unidades="4">silla cocina nogal</item>
    </op>
  - <res name="Order">
    - <order>
        <id>345</id>
        <item unidades="1" precio="123456">mesa comedor haya</item>
        <item unidades="4" precio="13457">silla cocina nogal</item>
        <total>177284</total>
      </order>
    </res>
    <op name="List" u="juan" pass="nauj" />
  - <res>
    - <order>
        <id>345</id>
        <item unidades="1" precio="123456">mesa comedor haya</item>
        <item unidades="4" precio="13457">silla cocina nogal</item>
        <total>177284</total>
      </order>
    - <order>
        <item unidades="0" precio="123456">mesa comedor haya</item>
      </order>
      <total>1234567</total>
    </res>
  </list>
```

En segundo lugar, modificar el ejemplo de contador con JDOM para leer un documento xml como `shop.xml` y que escriba en la salida en forma textual, la lista de pedidos con los datos que aparecen en el documento xml. Información adicional de JDOM puede encontrarse en el web de lab y en el web de jdom: `http://www.jdom.org`.