Sistemas Distribuidos

Tema 6:
XML en Sistemas Cliente/Servidor

E. U. Informática en Segovia

Departamento de Informática

Universidad de Valladolid

Sistemas Distribuidos

6.1 Introducción

Febrero de 1998: W3C (*World Wide Web Consortium*) crea una recomendación de un estándar para documentos en la web llamado eXtensible Markup Language o XML (*lenguaje de marcado extensible*).

Origen del XML: SGML ⇒ `aspecto` similar a HTML. En HTML las etiquetas (*tags*) predefinidas, en XML conjunto de etiquetas abierto ⇒ estructura del documento abierta.

En XML etiquetas incluídas entre ángulos <...>.

- HTML: etiquetas especifican cómo se va a mostrar la información.
- XML: etiquetas identifican los datos ⇒ actúan como el nombre de un campo. Se pueden usar *tags* con sentido para una aplicación determinada.

XML se está convirtiendo en un estándar para el intercambio de información en Internet. SOAP y XML-RPC (basados en XML) muy de moda para la obtención de información en Internet.

Sistemas Distribuidos

6.2 Estructura de XML

En XML la sintaxis es más restrictiva que en HTML. Ejemplo:

Válido en HTML, no válido en XML:

```
<ol>
   <li> Uno
   <li> Dos
</ol>
```

Válido en HTML y en XML:

```
<ol>
   <li> Uno </li>
   <li> Dos </li>
</ol>
```

Documento XML que cumple reglas generales y con etiquetas válidas:

documento XML bien formado (*well-formed*)

6.2.1. Prólogo

Prólogo mínimo XML:

```
<?xml version="1.0"?>
```

Prólogo XML con información adicional:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
```

Atributos de la declaración:

- version: identifica la versión de XML. Atributo obligatorio
- encoding: juego de caracteres utilizado. iso-8859-1 para Europa Occidental. Por defecto unicode: utf-8
- standalone: si yes puede haber referencias externas. Por defecto no

Además, declaraciones de entidades o especificaciones en una DTD (*Document Type Definition*):

```
<!DOCTYPE mensaje SYSTEM "mensaje.dtd">
```

6.2.2. Cuerpo del documento

Cuerpo: elementos entre *tag* de apertura y de cierre

```
<mensaje>
   <para>tu@tu.casa.es</para>
</mensaje>
```

Un *tag* puede contener otros ⇒ estructura jerárquica. Ej. para mensajes:

```
<mensaje>
   <para>tu@tu.casa.es</para>
   <de>yo@mi.casa.es</de>
   <asunto>El XML es una pasada</asunto>
   <texto>
      Pues eso, que el XML es una pasada y me gusta...
   </texto>
</mensaje>
```

Tag de apertura ⇒ *tag* de cierre. Ej. incorrecto:

```
<mensaje> <para>
   ...
</mensaje> </para>
```


Tag vacío: acaba en /> en vez de en >

```
<mensaje>
   <para>tu@tu.casa.es</para>
   <de>yo@mi.casa.es</de>
   <asunto>El XML es una pasada</asunto>
   <urgente/>
   <texto>
      Pues eso, que el XML es una pasada y me gusta...
   </texto>
</mensaje>
```

Tag con atributos para información adicional:

```
<mensaje para="tu@tu.casa.es" de="yo@mi.casa.es">
   <asunto>El XML es una pasada</asunto>
   <urgente/>
   <texto>
      Pues eso, que el XML es una pasada y me gusta...
   </texto>
</mensaje>
```

La DTD especifica tags válidos y sus atributos. Sin DTD, reglas genéricas

6.2.3. Otros elementos

Comentarios como en HTML:

```
<!-- Esto es un comentario -->
```

Órdenes de procesamiento:

```
<?aplicacion instrucciones?>
```

donde aplicacion es el procesador e instrucciones es información para la aplicacion

En el prólogo:

```
<?xml version="1.0"?>
```

es una instrucción de procesamiento más

6.3 Estructura de una DTD

Para

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE mensaje SYSTEM "mensaje.dtd">
<mensaje>
   <para>tu@tu.casa.es</para>
   <de>yo@mi.casa.es</de>
   <asunto>El XML es una pasada</asunto>
   <urgente/>
   <texto>
      Pues eso, que el XML es una pasada y me gusta...
   </texto>
</mensaje>
```

mensajes.dtd podría ser

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- DTD para mensajes -->
<!ELEMENT mensaje (para,de,asunto?,urgente?,texto?)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT asunto (#PCDATA)>
<!ELEMENT urgente EMPTY>
<!ELEMENT texto (#PCDATA)>
```

Calificadores: ? opcional * cero o más + uno o más , secuencia | opción

Ej: (para|de)* es un número indeterminado de para y de intercalados

Para

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE mensaje SYSTEM "mensaje.dtd">
<mensaje para="tu@tu.casa.es" de="yo@mi.casa.es">
   <asunto>El XML es una pasada</asunto>
   <urgente/>
   <texto>
      Pues eso, que el XML es una pasada y me gusta...
   </texto>
</mensaje>
```

mensajes.dtd podría ser

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- DTD para mensajes -->
<!ELEMENT mensaje (asunto?,urgente?,texto?)>
<!ATTLIST mensaje
    para CDATA #REQUIRED
    de   CDATA #REQUIRED
>
<!ELEMENT asunto (#PCDATA)>
<!ELEMENT urgente EMPTY>
<!ELEMENT texto (#PCDATA)>
```

Otras posibilidades:

- Atributos con conjunto de valores posibles restringido
- Atributos opcionales con valor por defecto
- Atributos con estructura
- Entidades (sustitución textual)

6.4 Utilización de XML

6.4.1. Ventajas de XML

- *Texto plano*: creado y editado con cualquier editor y alta portabilidad y facilidad de transmisión en distinto hardware o sistemas operativos
- *Identificación de datos*: XML dice el tipo de datos, no cómo mostrarlos. Ejemplo de mensajes: programa de correo para procesar, programa de agenda para extraer remitentes y programa de búsqueda para buscar mensajes con determinados criterios
- *Estilos de presentación*: para mostrar los datos o convertirlos hace falta algún tipo de hoja de estilos ⇒ alta flexibilidad
- *Jerarquía de los datos*: ventajas de la estructura jerárquica de XML:
  - Estructuras jerárquicas de acceso más rápido que las secuenciales
  - Reordenación de datos más sencilla: relocalización de datos
  - Serialización resuelta

6.4.2. Usos de XML

- *Procesamiento de datos*: XML se está conviertiendo en el estándar de representación de datos en la Web
- *Programación orientada a documento*: se utiliza XML para construir documentos que describen cómo una aplicación deba comportarse. Por ejemplo, *VoiceXML*, que se utiliza para la construcción de sistemas de diálogo.
- *Enlace*: el uso de las DTD asegura la compatibilidad para las aplicaciones
- *Archivo*: XML es un formato ideal para el archivo de datos. Al ser texto se asegura la portabilidad

6.5 Acceso a ficheros XML

Para procesar XML hay librerías de procesamiento en gran cantidad de lenguajes de programación (*C*, *C++*, *Java*, *Perl*, *Python*...)

Dos formas generales de procesar XML:

- APIs de acceso secuencial o SAX (*Serial Access for XML o Simple Api for XML*)
- APIs de acceso en árbol o DOM *(Document Object Model)*

En ambos casos, parser automático: se carga el fichero XML, se ve si hay DTD asociada y si algo no se ajusta a la DTD ⇒ error

Además, cosas adicionales como XSLT o XPath

6.5.1. Acceso secuencial: SAX

APIs SAX *orientadas a eventos* el XML se va procesando y se generan eventos cuando se encuentran etiquetas, atributos, datos...

Ventajas de SAX:

- Comienzo inmediato y requerimientos de memoria reducidos
- Se tratan los eventos que se quiera. Ej.: si se busca información de las direcciones de los mensajes, se trata solamente eso.

Desventaja de SAX:

- No hay almacenamiento de los nodos

6.5.2. Acceso en árbol: DOM

APIs DOM orientadas a árboles: se obtiene una estructura de objetos que representa la estructura del documento. Se accede al nodo raíz y la API ofrece métodos para acceder a los nodos hijos de cada nodo y a sus características

Ventaja de DOM:

- Documento disponible completo, por lo que cualquier aplicación puede navegar por todo el documento

Desventajas de DOM:

- Documento cargado completo en memoria ⇒ problema si el tamaño del documento es considerable
- Si se busca algún nodo concreto hay que navegar por todo el árbol y despreciar el resto

6.5.3. XSLT

XSLT = *XML Stylesheet Language for Transformations*. Especifica el estilo para la transformación de ficheros XML. Puede usarse para generar PDF, postscript o HTML (lo habitual)

Similar a los ficheros css en HTML

6.5.4. XPath

Lenguaje de especificación para trabajar con XSLT. Permite especificar caminos en la estructura de XML. Por ejemplo diferencia <title> en:

```
<article>
   <title>
      ...
   </title>
</article>
```

```
<person>
   <title>
      ...
   </title>
</person>
```


6.6 Ejemplo de programación con SAX

Objetivo: extraer una lista de los destinatarios de un conjunto de mensajes de correo electrónico

```
<!-- mensajes.xml -->
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE mensajes SYSTEM "mensajes.dtd">

<mensajes>

<mensaje>
   <para alias="tumismo">tu@tu.casa.es</para>
   <de>yo@mi.casa.es</de>
   <asunto>El XML es una pasada</asunto>
   <urgente/>
   <texto>
      Pues eso, que el XML es una pasada y me gusta...
   </texto>
</mensaje>

<mensaje>
   <para>otrotu@su.casa.es</para>
   <de>yo@mi.casa.es</de>
   <asunto>
      En XML se puede hacer de todo
   </asunto>
   <urgente/>
   <texto>
      Se puede procesar con SAX o DOM
   </texto>
</mensaje>

</mensajes>
```

```
<!-- mensajes.xml -->
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- DTD para mensajes -->
<!ELEMENT mensajes (mensaje+)>
<!ELEMENT mensaje (para,de,asunto?,urgente?,texto?)>
<!ELEMENT para (#PCDATA)>
<!ATTLIST para
   alias CDATA "">
<!ELEMENT de (#PCDATA)>
<!ELEMENT asunto (#PCDATA)>
<!ELEMENT urgente EMPTY>
<!ELEMENT texto (#PCDATA)>
```

```
// ExtraePara.java
import java.io.*;
import java.util.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;  // Parser XML

// Class ExtraePara

public class ExtraePara extends DefaultHandler {

   static boolean leyendoPara = false;
   String cadenaPara="";
   String cadenaAlias="";

   // Método main

   public static void main(String argv[]) {
      if (argv.length == 0) {
         System.err.println("Java ExtraePara fichero");
         System.exit(1); }

         // Use an instance of ourselves as the SAX event handler
         DefaultHandler handler = new ExtraePara();

         // Use the default (non-validating) parser
         SAXParserFactory factory = SAXParserFactory.newInstance();
         factory.setValidating(true);
         try { // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler);
         } catch (Exception e) {
            e.printStackTrace();
         }
   } // ExtraePara.main
   ....
```

Configura el parser con validación

Hace el parsing del fichero de entrada


```
// ExtraePara.java (cont.)
   ....
   // Métodos manejadores de eventos SAX

   public void setDocumentLocator(Locator l) {
      System.out.println("Fichero: " + l.getSystemId()); }
   public void startDocument() {
      System.out.println("Principio de documento"); }
   public void endDocument() {
      System.out.println("Fin del documento"); }

   // Comienzo de elementos

   public void startElement(String namespaceURI,
         String lName, String qName,
         Attributes attrs) throws SAXException {

      String eName = lName; // nombre del elemento
      if ("".equals(eName)) eName = qName;

      // Principio de para
      if ("para".equals(eName)) {
         cadenaPara="";
         cadenaAlias="";
         leyendoPara=true;
         for (int i = 0; i < attrs.getLength(); i++) {
            String aName = attrs.getLocalName(i); // Attr name
            if ("".equals(aName))
               aName = attrs.getQName(i);
            if ("alias".equals(aName)) {
               cadenaAlias=attrs.getValue(i);
            }
         }
      }
   ....
```

Principio de parsing
Principio de documento
Final de documento

Comienzo de elementos

Se ejecuta al encontrar <para>

```
// ExtraePara.java (final)
   ....
   // Final de elementos

   public void endElement(String namespaceURI, String sName,
         String qName) throws SAXException {
      String eName = sName; // nombre del elemento
      if ("".equals(eName)) eName = qName;

      // Final de para
      if ("para".equals(eName)) {
         leyendoPara = false;
         System.out.print("Destinatario: "+cadenaPara);
         if (!("".equals(cadenaAlias)))
            System.out.print(" ("+cadenaAlias+")");
         System.out.println(); }
   }

   // Lector de cadenas de caracteres

   public void characters(char buf[], int offset, int len) {
      String s = new String(buf, offset, len);
      if (leyendoPara) cadenaPara = cadenaPara+s;
   }

   // Manejadores de errores no fatales

   public void error(SAXParseException e) throws SAXParseException {
      throw e; }

   public void warning(SAXParseException err) throws SAXParseException {
      System.out.println("**** Warning"
            + ", línea " + err.getLineNumber()
            + ", uri " + err.getSystemId() + "****");
      System.out.println("    " + err.getMessage()); }

} // class ExtraePara
```

Final de elementos

Se ejecuta al encontrar </para>

Se ejecuta dentro de cualquier tag

Resultado de la ejecución:

```
$ java ExtraePara mensajes.xml
Fichero: file:/home/leandro/tmp/mensajes.xml
Principio de documento
Destinatario: tu@tu.casa.es (tumismo)
Destinatario: otrotu@su.casa.es
Fin del documento
$
```

Varias cosas destacables:

- main es siempre similar. Para probar si el XML bien formado:
  factory.setValidating(true);
- startElement y endElement invocados cada vez que comienza y acaba un elemento.
- characters invocado al leer el contenido de una etiqueta
- Argumentos de etiquetas: en startElement a partir del objeto de la clase Attributes
- alias siempre tiene valor (por defecto la cadena vacía)

6.7 Ejemplo de programación con DOM

```
// ExtraePara.java
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import java.io.*;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

// Class ExtraePara

public class ExtraePara {

   static Document document;

   // Método main

   public static void main(String argv[]) {
      NodeList listaNodos;

      if (argv.length != 1) {
         System.err.println(
               "Uso: java ExtraePara filename");
         System.exit(1);
      }

      DocumentBuilderFactory factory =
         DocumentBuilderFactory.newInstance();
      factory.setValidating(true);
      factory.setNamespaceAware(true);
      ....
```

El objetivo el mismo: extraer una lista de los destinatarios de un conjunto de mensajes de correo electrónico

Objeto para acceder al árbol

Para acceder a todos los nodos <para>

Activa comprobación por dtd


```
// ExtraePara.java (cont.)
      ....
      try {
         DocumentBuilder builder = factory.newDocumentBuilder();
         document = builder.parse( new File(argv[0]) );

         listaNodos = document.getElementsByTagName("para");

         for (int i=0 ; i<listaNodos.getLength() ; i++) {

            //System.out.println("Nodo: "+i);
            Node nodo;
            nodo = listaNodos.item(i);

            Node subNodo;
            subNodo = nodo.getFirstChild();
            System.out.print(" "+subNodo.getNodeValue());

            NamedNodeMap atts;
            atts = nodo.getAttributes();
            //System.out.println("  Atributos: "+atts.getLength());

            Node atributo;
            atributo = atts.item(0);
            String alias = atributo.getNodeValue();
            if (!"".equals(alias))
               System.out.print(" ("+alias+")");
            System.out.println();

         }

      } catch (Exception e) {
         e.printStackTrace();
      }
   }// main

}// ExtraePara
```

Obtiene el árbol

Obtiene los nodos <para>

Hay un nodo intermedio que contiene el valor del nodo

Se accede al primer atributo

Resultado de la ejecución:

```
$ javac ExtraePara.java
$ java Extraepara mensajes.xml
 tu@tu.casa.es (tumismo)
 otrotu@su.casa.es
$
```

Varias cosas destacables:

- main es siempre similar. Para probar si el XML bien formado:
  factory.setValidating(true);
- Se almacena en memoria todo como un árbol en el cual todo son nodos. Incluso los atributos de una etiqueta son a su vez nodos
- El árbol hay que recorrerlo, aunque hay métodos como getElementsByTagName que implementan funciones de búsqueda.
- El atributo alias siempre tiene valor (por defecto la cadena vacía)

Otro ejemplo: seleccionar un mensaje por índice y mostrarlo

```
// ExtraeMensaje.java
import javax.xml.parsers.DocumentBuilder;
  ... aquí un montón de líneas import
import org.w3c.dom.DOMException;

// Class ExtraeMensaje

public class ExtraeMensaje {

   static Document document;

   // Método main

   public static void main(String argv[]) {
      NodeList listaNodos;

      if (argv.length != 2) {
         System.err.println(
               "Uso: java ExtraeMensaje filename indice");
         System.exit(1);
      }

      Integer indInteger = new Integer(argv[1]);
      int indiceMensaje = indInteger.intValue()-1;

      if (indiceMensaje < 0)
      {
         System.out.println("ERROR");
         System.exit(1);
      }

      System.out.println("Extrayendo mensaje "+(indiceMensaje+1));
      ....
```

Objeto para acceder al árbol

Para acceder a todos los nodos <mensaje>

```
// ExtraeMensaje.java (cont.)
      ....
      DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
      factory.setValidating(true);
      factory.setNamespaceAware(true);

      try {
         DocumentBuilder builder = factory.newDocumentBuilder();
         document = builder.parse( new File(argv[0]) );

         listaNodos = document.getElementsByTagName("mensaje");
         if (indiceMensaje > listaNodos.getLength()-1) {
            System.out.println("No hay tantos mensajes");
            System.exit(1); }

         Node nodo;
         nodo = listaNodos.item(indiceMensaje);

          // Use a Transformer for output
          TransformerFactory tFactory =  TransformerFactory.newInstance();
          Transformer transformer = tFactory.newTransformer();

          DOMSource source = new DOMSource(nodo);
          StreamResult result = new StreamResult(System.out);

          if (document.getDoctype() != null){
             String systemValue = (new File(document.getDoctype().getSystemId())).getName();
             transformer.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM, systemValue);
          }
          transformer.transform(source, result);

          System.out.println();

      } catch (Exception e) {
         e.printStackTrace();
      }
   }// main

}// ExtraeMensaje
```

Obtiene el árbol del fichero de entrada

Obtiene el mensaje especificado

Crea un nuevo árbol DOM

Usa la dtd del fichero original

Genera el árbol DOM como texto

Resultado de la ejecución:

```
$ java ExtraeMensaje mensajes.xml 1
Extrayendo mensaje 1
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mensaje SYSTEM "mensajes.dtd">
<mensaje>
   <para alias="tumismo">tu@tu.casa.es</para>
   <de>yo@mi.casa.es</de>
   <asunto>El XML es una pasada</asunto>
   <urgente/>
   <texto>
      Pues eso, que el XML es una pasada y me gusta...
   </texto>
</mensaje>
$ java ExtraeMensaje mensajes.xml 4
Extrayendo mensaje 4
No hay tantos mensajes
$
```

Varias cosas destacables:

- Se selecciona el nodo deseado con el método item() de la clase Document
- Se crea un nuevo árbol dom mediante el constructor de la clase DOMSource
- Se utiliza la clase Transformer para lograr la salida del mensaje. Se utilizan el subárbol anteriormente seleccionado y la salida va a System.out

6.8 Ejemplo de programación con XSLT

Para transformar con XSLT, se debe utilizar la API DOM

Al crear la instancia de la clase Transformer, se especifica el fichero XSLT. La transformación está especificada completamente en el fichero XSLT, lo que tiene dos ventajas:

- No hace falta recompilar cuando se cambia la transformación
- El mismo ejecutable vale para distintas transformaciones


```
// Procesa.java
// Aquí van un montón de líneas import
public class Procesa {

  static Document document;

  public static void main (String argv []) {

    if (argv.length != 2) {
      System.err.println ("Uso: java Procesa stylesheet xmlfile");
      System.exit (1);
    }

    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

    try {
      File stylesheet = new File(argv[0]);
      File datafile  = new File(argv[1]);

      DocumentBuilder builder = factory.newDocumentBuilder();
      document = builder.parse(datafile);

      // Use a Transformer for output
      TransformerFactory tFactory = TransformerFactory.newInstance();
      StreamSource stylesource = new StreamSource(stylesheet);
      Transformer transformer = tFactory.newTransformer(stylesource);

      DOMSource source = new DOMSource(document);
      StreamResult result = new StreamResult(System.out);
      transformer.transform(source, result);

    } catch (Exception e) {
      e.printStackTrace();
    }

  } // main
}
```

Aquí se especifica la transformación

```
<?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"/>

   <xsl:template match="/">
      <html>
         <body>
           <h1>Lista de Mensajes</h1>
            <ul>
               <xsl:apply-templates/>
            </ul>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="/mensajes/mensaje">
      <li>
         <xsl:apply-templates/>
      </li>
      <br/>
   </xsl:template>

   <xsl:template match="mensaje/para">
      <b>Para: </b>
         <font color="red">
           <xsl:apply-templates/>
         </font>
   </xsl:template>

   <xsl:template match="mensaje/de">
      <br/><b>De: </b>
         <font color="blue">
            <xsl:apply-templates/>
         </font>
   </xsl:template>
```

```
   <xsl:template match="mensaje/asunto">
      <br/><b>Asunto: </b>
         <xsl:apply-templates/>
   </xsl:template>

   <xsl:template match="mensaje/texto">
      <br/><b>Texto: </b>
      <i>
         <xsl:apply-templates/>
      </i>
   </xsl:template>

</xsl:stylesheet>
```

- Se especifica que la salida va a ser en html ⇒ no encabezado XML, líneas en blanco, etc
- Procedimiento recursivo. / es la raíz del documento, y a partir de ella estructura como de caminos a directorios
- <xsl:apply-templates/> especifica que se apliquen las transformaciones al contenido de lo que ha coincidido en el momento en el que se encuentran


```
<html>
<body>
<h1>Lista de Mensajes</h1>
<ul>


<li>

<b>Para: </b><font color="red">tu@tu.casa.es</font>

<br>
<b>De: </b><font color="blue">yo@mi.casa.es</font>

<br>
<b>Asunto: </b>El XML es una pasada

   <br>
<b>Texto: </b><i>
      Pues eso, que el XML es una pasada y me gusta...
   </i>

</li>
<br>

....

</ul>
</body>
</html>
```

Resultado de la ejecución:

```
$ java Procesa xml2html.xsl mensajes.xml > mensajes.html
$
```

Lista de Mensajes

- **Para:** tu@tu.casa.es
  De: yo@mi.casa.es
  Asunto: El XML es una pasada
  Texto: *Pues eso, que el XML es una pasada y me gusta...*

- **Para:** otrotu@su.casa.es
  De: yo@mi.casa.es
  Asunto: En XML se puede hacer de todo
  Texto: *Se puede procesar con SAX o DOM*

Otro ejemplo: extracción de las direcciones de los destinatarios:

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<xsl:output method="xml" indent="yes"/>

<xsl:template match="/">
   <xsl:copy-of select="mensajes/mensaje/para"/>
</xsl:template>

</xsl:stylesheet>
```

```
$ java Procesa extraepara.xsl mensajes.xml
<?xml version="1.0" encoding="UTF-8"?>
<para alias="tumismo">tu@tu.casa.es</para>
<para alias="">otrotu@su.casa.es</para>
$
```

Observaciones:

- En la segunda dirección, alias presenta el valor por defecto de la dtd (“”)
- <xsl:output method="xml" indent="yes"/> provoca la aparición del encabezado xml, espacios en blanco e indentación
- <xsl:copy-of select="mensajes/mensaje/para"/> especifica que se copian las etiquetas <para>
- El fichero xml generado ya no tiene dtd asociada

Para extraer las etiquetas <de>

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<xsl:output method="xml" indent="yes"/>

<xsl:template match="/">
   <xsl:copy-of select="mensajes/mensaje/de"/>
</xsl:template>

</xsl:stylesheet>
```

```
$ java Procesa extraede.xsl mensajes.xml
<?xml version="1.0" encoding="UTF-8"?>
<de>yo@mi.casa.es</de>
<de>yo@mi.casa.es</de>
$
```

Otras funcionalidades de XSLT:

- Seleccionar y manipular atributos de etiquetas
- Utilizar variables internas
- Realizar procesamientos condicionales y bucles
- Utilizar funciones numéricas, booleanas, de cadenas...

6.9 SOAP y servicios web

6.9.1. Introducción

IMR: el objetivo es invocar de forma similar a local. Vimos SunRPC, JavaRMI y CORBA.

Inconvenientes de estas alternativas:

- Cada una su protocolo y sus tipos de datos ⇒ poca interoperabilidad. Problemas de interacción entre implementaciones de CORBA
- Firewalls en redes y máquinas ⇒ bloqueo de puertos. A menudo solamente disponibles los puertos típicos para usar en Internet.

Con la explosión de la Web, la tendencia es a:

- Independencia de la plataforma
- Protocolos estándar de comunicación (HTTP...)
- Protocolos estándar de descripción de información (XML)

con lo que es lógico que se busque esto en la IMR.

Primera alternativa: XML-RPC. Consiste en usar XML en solicitud y respuesta. Ventajas:

- XML es ASCII ⇒ no hay problemas en el transporte y evitamos tener un nuevo puerto abierto
- Cualquier lenguaje de programación vale para implementación

Sobre esta base se desarrolló SOAP (*Simple Object Access Protocol*), que refina XML-RPC con mayor flexibilidad, extensibilidad y mayor número de herramientas de desarrollo

Por tanto, SOAP:

- Es un protocolo ligero de comunicación entre aplicaciones
- Está diseñado para comunicarse vía http (admite otros protocolos)
- No está sujeto a ninguna tecnología concreta
- Es independiente de los lenguajes de programación
- Se basa en XML y es una propuesta de la W3C ⇒ estándar

6.9.2. Servicios web

Servicio Web: módulo de software (en cualquier lenguaje) identificado por una URI (*Uniform Resource Identifier*), que ofrece una interfaz pública a través de Internet, mediante peticiones codificadas en un protocolo de invocación remota basado en XML, (XML-RPC o SOAP)

En la práctica, se usa SOAP ⇒ *servicios Web = servicios basados en SOAP*

Se accede al servicio usando HTTP y se implementa con tecnologías web (CGI, servlets, ASP, PHP, ISAPI, NSAPI) ⇒ un servidor de páginas puede ser servidor de servicios web

Arquitectura de servicios web (de abajo arriba):

- Los servicios de red: *sockets*, *HTTP*, *FTP*...
- Los servicios de mensajes e invocación remota: *XML-RPC*, *SOAP*...
- Capa de servicios que proporcionan descripción de la interfaz de los servicios inferiores en formato estándar. Esto está soportado por el protocolo WSDL (*Web Service Description Language*), basado en XML
- Capa que proporciona servicios de localización y publicación de servicios Web. Se basa en el protocolo UDDI (*Universal Description, Discovery and Integration*)

6.9.3. Diferencias de SOAP con RMI, CORBA...

- *SOAP es un protocolo*, no un modelo ni plataforma.
- Filosofía: publicar la información o servicios de páginas Web, en un formato válido para las aplicaciones (no para las personas)
- JavaRMI o CORBA incorporan, localización de recursos, alineado/aplanado y desalineado/aplanado , lenguajes de descripción de interfaces.... SOAP no. Con WSDL y UDDI hay descripción de interfaz y servicios de descubrimiento.

SOAP gana en:

- Sencillez
- Independencia de plataforma
- Uso de infraestructuras web

6.9.4. Invocación remota en SOAP

Puntos básicos de la definición de SOAP:

- Los mensajes deben ir contenidos en un sobre o envoltura (envelope) que define un marco para describir qué hay en un mensaje, qué estructura debe tener y como procesarlo
- Hay unas reglas de codificación para describir instancias de tipos de datos definidos por las aplicaciones
- Hay una convención para representar llamadas a procedimientos remotos y sus respuestas

Mensaje SOAP = documento XML bien formado. Partes o elementos:

- Sobre, (Envelope): elemento obligatorio de mayor nivel. Puede contener atributos y subelementos
- Cabecera (Header), opcional, y que será el primer hijo directo de Envelope. Uso: pasar información adicional entre aplicaciones para lo que puede definir elementos que especifiquen valores
- Cuerpo (Body), obligatorio e hijo directo de Envelope, después de Header. En él se define un elemento Fault opcional para indicar mensajes de error
- No pueden haber referencias a DTD's ni instrucciones de procesamiento
- Usa *XML Schema Language* para los tipos de datos

Ejemplo: obtención de la provincia por el código de dos cifras

Solicitud:

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Header> </SOAP-ENV:Header>
  <SOAP-ENV:Body>
   <ns1:getNombreProvincia
     SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <name xsi:type="xsd:int">36</name>
     </ns1:getNombreProvincia>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Respuesta:

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Header> </SOAP-ENV:Header>
     <SOAP-ENV:Body>
       <ns1: getNombreProvinciaResponse
         SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
          <return xsi:type="xsd:string">Pontevedra</return>
       </ns1: getNombreProvinciaResponse>
     </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Comentarios:

- El cliente invoca por URI ⇒ no hay suplentes
- No objetos como parámetros ⇒ no hay posibilidad de retrollamadas
- Los servicios siempre no persistentes
- *Espíritu web*: solicitudes autocontenidas e indendientes sin creación de objetos en el servidor

Para implementar persistencia, dos posibilidades:

- El cliente almacena un estado, y lo envía al servidor en Header. El servidor responde con el nuevo estado. Es algo parecido a las *cookies*
- Implementar mecanismos para que el cliente active objetos, los identifique en las solicitudes y ordene al servidor almacenarlos. No tiene sentido

Pasos de la invocación remota usando SOAP:

- Un cliente envía a un servidor Web un mensaje *HTTP POST* con la URI destino de los datos. Similar a *POST* para formularios, pero los datos son la petición de invocación remota en SOAP
- El servidor Web trata esta petición como habitualmente: ejecuta la rutina asociada con la URI, pasándole los datos como parámetro
- La rutina entiende la petición SOAP, ejecuta el método solicitado con los parámetros, y devuelve el resultado como respuesta SOAP
- El servidor Web genera la respuesta en *HTTP* en la que incrusta los datos devueltos por la rutina (igual que para páginas Web)
- El cliente recibe la respuesta e interpreta los datos en SOAP

6.9.5. WSDL y UDDI

WSDL

El protocolo WSDL (*Web Service Description Language*) viene a jugar el papel de un IDL (*Interface Description Language*) para los servicios Web. Diferencia: se suele hacer después de implementar el servidor:

- Herramientas que lo generan a partir de la interfaz de la implementación
- No es fácilmente entendible y manipulable

Diferencia WSDL con IDL clásicos:

- IDL describe la interfaz. WSDL describe más cosas (información sobre el protocolo y dirección del servicio Web). Así los clientes pueden importar los datos relativos a la conexión

Se dice que WSDL describe lo abstracto (la interfaz del servicio) y lo concreto (detalles de implementación y localización) y los relaciona. Los IDL clásicos se limitan a lo abstracto

Descripción servicio de traducción *babelfish*:

```
http://www.xmethods.net/sd/2001/BabelFishService.wsdl
```

Implementación del servicio:

```
http://services.xmethods.net:80/perl/soaplite.cgi
```

UDDI

UDDI (*Universal Description, Discovery and Integration*) especifica una estructura de registro para la publicación y localicación de servicios Web

La idea es proporcionar un directorio compartido, a modo de páginas amarillas, donde se expone información de los servicios y sus interfaces

Por ejemplo, que las operadoras telefónicas registren en UDDI sus servicios de búsqueda de números de abonados

UDDI también puede usarse de forma privada

6.9.6. Un ejemplo

Servidor que recibe una cadena y devuelve un saludo

Requisito: tener un servidor funcionando (por ejemplo, *Tomcat*). Podemos ver los servicios SOAP que despliega:

```
http://localhost:8080/soap/servlet/rpcrouter
```

Programa servidor:

```
import java.util.*;
import org.apache.soap.util.xml.*;

public class HolaServer
{
  public String DiHola(String nombre)
{
    String ret;
    ret = "Hola " + nombre + "!!";
    return ret;
  }
}
```

Para *desplegarlo*, creamos descriptor.xml y lo registramos en el Tomcat:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
      id="urn:HolaServer">
  <!-- Definicion de los metodos de la clase -->
  <isd:provider type="java" scope="Request" methods="DiHola">
    <!-- Definicion de la clase -->
    <isd:java class="HolaServer" static="false"/>
  </isd:provider>
</isd:service>
```

```
// varios import

public class HolaCliente {
  public static void main(String[] args) throws Exception {

    URL url = new URL("http://localhost:8080/soap/servlet/rpcrouter");

    Call call = new Call();
    call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
    call.setTargetObjectURI("urn:HolaServer");
    call.setMethodName("DiHola");

    Vector params = new Vector();
    params.addElement(new Parameter("nombre", String.class, "Juan", null));
    call.setParams(params);

    Response resp;
    try    {
          resp = call.invoke(url, "");
    }    catch (SOAPException e) {
      System.err.println("Capturada SOAPException");
      System.err.println(" Código: " + e.getFaultCode());
      System.err.println(" Mensaje: " + e.getMessage());
      return;
    }
  }
}
```