Capítulo 4. Implementación

En esta sección se detallan las herramientas implementadas con base en el diseño realizado, así como las clases implementadas para lograr los objetivos planteados.

4.1 Herramientas implementadas

Las herramientas implementadas para cada etapa del proceso de conversión, se ilustran en la Figura 4.1, la cual corresponde directamente al diseño propuesto en la Figura 3.3.

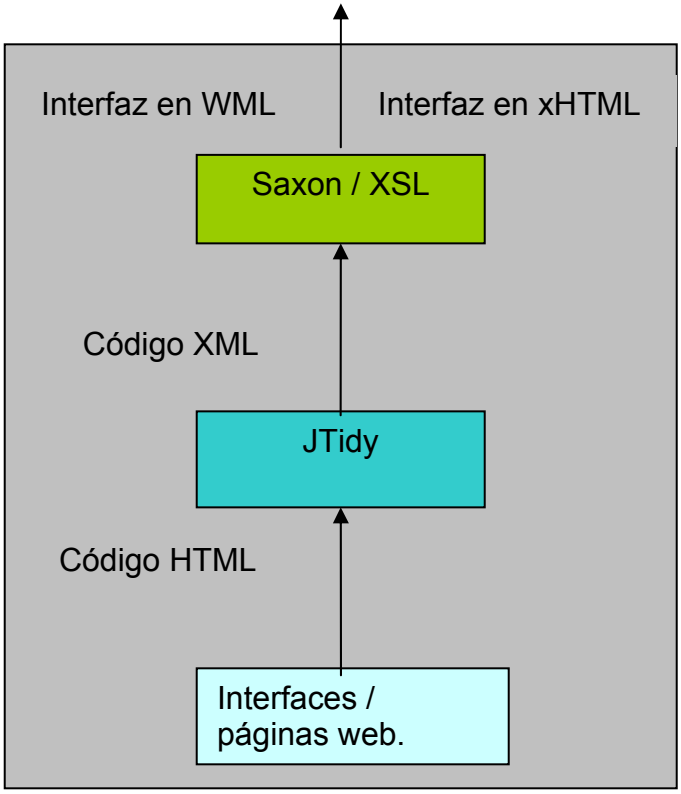

Figura 4.1 Herramientas utilizadas en la aplicación EditMos

Para la depuración del documento en HTML se utilizó JTidy, herramienta descrita en la sección 2.2. Al resultado se le aplican etiquetas convenientes para su transformación al lenguaje XML, siendo este lenguaje el lenguaje intermedio para la transformación. Este componente fue agregado con el fin de que el administrador, contrario al caso de PoPS, no se vea en la necesidad de manipular XML manualmente. La aplicación se encargará de manejar el documento HTML y generar el documento XML correspondiente con la mínima interacción del usuario en cuanto a qué etiqueta en lenguaje HTML es equivalente en el lenguaje XML. Teniendo el documento XML bien formado y válido se podrá hacer uso de éste en conjunto con las hojas de estilo y Saxon, un procesador XML basado en SAX para la transformación pertinente. Se consideraron las ventajas de utilizar la herramienta descrita en la sección 2.4, pero no permitía la extensibilidad de la herramienta al tener hojas de estilo predefinidas que no se podían extender. El proceso no es muy transparente como para poder integrarlo finalmente a la aplicación a desarrollar. En vista de esto, y ya que la herramienta se basa en las transformaciones XSLT, es decir, de un analizador con hojas de estilo, se decidió manejar la misma mecánica de PoPS, el uso de un analizador: Saxon, que está basado en la tecnología SAX (descrita en la Sección 2.5). Con esto, la extensibilidad de la aplicación no se limita, ya que con solo agregar una hoja de estilo (plantilla), se podrá convertir un archivo determinado en otro con la estructura que se le indica en la hoja de estilo. Las trasformaciones actuales se basan en archivos XML que concentran el contenido mientras que las hojas XSL concentran la estructura de cómo se desplegará la información del archivo XML.

4.2 Clases implementadas

A continuación se explican brevemente las clases principales utilizadas para la elaboración de la aplicación y los diagramas de clase se muestran a detalle en el Apéndice B.

EditMos: es la clase principal que carga todo la interfaz principal (*frame)* y sus componentes, dando inicio a la interacción con el usuario después de que esté haya iniciado sesión como usuario registrado. Es la interfaz principal que despliega la aplicación con la que interactúa el usuario.

ETidy: es la clase que maneja la interfaz Tidy, un API en JAVA que permite limpiar el HTML y arregla etiquetas mal cerradas, vacías y elementos mal anidados entre otros. Recibe un URL y permite validar código HTML.

ManejoArchivo: es la clase que se encarga del manejo de archivos en cuanto a leer, y estructurarlo como archivo XML, agregándole las etiquetas que lo definen como tal.

ManejoVector: esta clase recibe el XML limpio, es decir, bien formado para que a continuación la clase SaxParse analice el documento y la extracción de los elementos y atributos se va realizando a la par del análisis.

ManejoListas: esta clase recibe los vectores con los elementos y atributos que ManejoVector se ha encargado de llenar para que ManejoListas pueda crear un árbol con la

estructura del archivo XML, así como la lista de elementos y atributos que se despliega en los páneles para manipulación del usuario.

SaxParse: esta clase implementa la interfaz DefaultHandler, una interfaz de SAX que es la más importante y la más usada ya que es llamada cada vez que se encuentra un elemento. Para implementar una interfaz, una clase debe proveer código para cada uno de los métodos definidos en la interfaz, estos se pueden apreciar en el diagrama correspondiente (ver Apéndice B).

SaxonParse: esta clase implementa las interfaces de SAXON (API basado en SAX) y permite hacer el análisis de un archivo XML en conjunto con una hoja de estilo que define el lenguaje final de conversión.

4.3 Manejo de archivos de entrada

Una vez explicados las clases que se implementaron para la conversión y conociendo la interfaz principal (Sección 3.2.3), se procede a explicar el contenido que se vacía en cada uno de los componentes de la interfaz ilustrando el resultado paso a paso del proceso completo de conversión.

Se inicia el proceso cuando el usuario proporciona un URL de la interfaz que desea convertir, como se observa en la Figura 4.2.

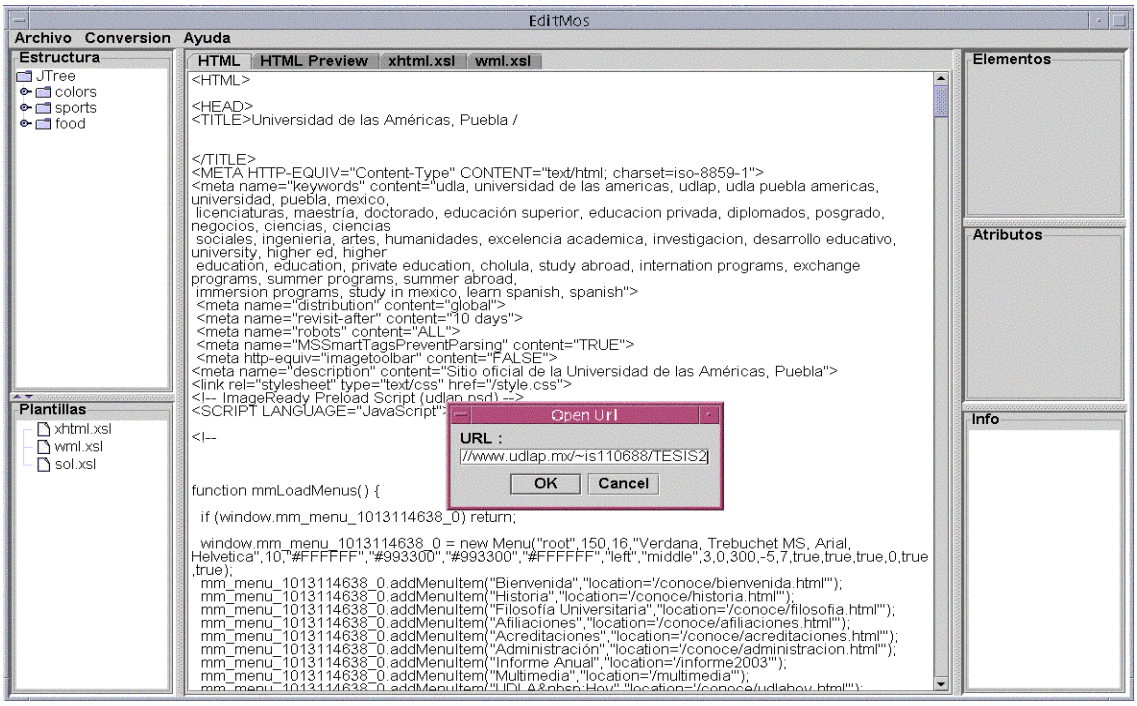

Figura 4.2 Acción: Abrir un URL

Si el usuario desea que el proceso sea en un solo paso, la aplicación manda el URL dado inmediatamente a limpiar, tratando de darle la mejor estructura para poder convertirlo a XML en la siguiente etapa. Para este fin, se utilizó el API de Tidy (ver Sección 2.3.3. Esto es porque el lenguaje de marcado HTML tiene la desventaja de que los navegadores lo puedan desplegar a pesar de estar mal estructurado (etiquetas finales faltantes o vacías), contrario a XML que debe conformar un documento válido y bien formado [Maruyama, 1999].

Un explorador HTML puede interpretar un documento HTML cuando un elemento no tiene etiquetas de cierre porque conoce la estructura del documento, sabe en qué lugar están permitidos los elementos y puede deducir dónde debe terminar un elemento.

Por lo tanto, con el siguiente fragmento de código HTML, un explorador no necesita etiquetas de cierre para los párrafos <p>, ni tampoco una etiqueta vacía para el salto.

```
<P><B> David Espinosa </B>
<P> Sta. Cruz Guadalupe <BR>
Puebla, PUE 72000 <BR>
MEX
<P> Tel: 2845899
<P> E-mail: david@micorreo.com
```

Listado 4.1 Un documento HTML no necesita etiquetas de cierre

El explorador puede deducir en dónde termina el párrafo porque sabe que los párrafos no deben estar anidados. Por lo tanto, el inicio de un nuevo párrafo debe coincidir con el final anterior. El explorador llena entonces los espacios en blanco e interpreta como:

```
<P><B> David Espinosa </B> </P>
<P> Sta. Cruz Guadalupe <BR> </BR>
Puebla, PUE 72000 <BR> </BR>
MEX </P>
<P> Tel: 2845899 </P>
<P> E-mail: david@micorreo.com </P>
```

Listado 4.2 El explorador interpreta el HTML

Sin embargo, un procesador XML no conoce la estructura del documento porque el usuario define sus propias etiquetas. Por lo tanto, un procesador XML no sabe que los elementos <p> (tampoco sabe que son párrafos) no pueden estar anidados. Si el Listado 4.1 fuera XML, el procesador lo interpretaría como se muestra a continuación:

```
<P><B> David Espinosa </B>
        <P> Sta. Cruz Guadalupe
                <BR> Puebla, PUE 72000 </BR>
                <BR> MEX </BR>
                <P> Tel: 2845899
                <P> E-mail: david@micorreo.com </P>
                </P>
        </P>
</P>
```

o como:

```
<P><B> David Espinosa </B> </P>
<P> Sta. Cruz Guadalupe
        <BR> Puebla, PUE 72000 </BR>
        </BR > MEX
        <P> Tel: 2845899 </P>
</P>
<P> E-mail: david@micorreo.com </P>
```

Listado 4.3 Diferentes interpretaciones del XML

Existen muchas otras posibilidades y ése es precisamente el problema. El procesador no sabría cuál elegir, por lo que el marcado debe ser claro. Y es debido a esto que el usuario tiene que intervenir en el proceso, ya que debe avisarle a la aplicación dónde desea las etiquetas de cierre, por lo cual se manejan las listas de elementos y atributos en los paneles de la izquierda, bajo la etiqueta de "Elementos" y "Atributos".

Validez

Un documento XML válido significa que el documento debe seguir las restricciones de validez (RV) especificadas en la Recomendación XML 1.0. Para que un documento sea revisado por validez, debe incluir la declaración <!DOCTYPE> al inicio del documento XML, que especifica el DTD de acuerdo al cual el documento debe ser validado. Las RV se enfocan a la estructura lógica de los elementos. Esto es, requieren que todas las etiquetas estén definidas en el DTD y que todos los elementos anidados a su vez, sigan el DTD, así como los atributos, etc.

Documentos bien formados

Esto se refiere a que el documento XML siga las restricciones de documento bien formado que se encuentran en la Recomendación 1.0. [Maruyama 2000]. Mientras la validez se encarga de la estructura lógica de los elementos, el término de "bien formado" se enfoca a la estructura física, como lo es la concordancia de las etiquetas. En XML cada etiqueta de

inicio, como por ejemplo <TR> debe tener su etiqueta de cierre, en este caso </TR>. De lo contrario, una etiqueta debe ser vacía y se representaría como <etiqueta/>. Esto puede parecer trivial pero es necesario puesto que XML no tiene elementos predefinidos.

Todos los procesadores XML reportan errores a las aplicaciones si el documento XML a analizar no esta bien formado. No debe haber errores como lo son la falta de etiquetas de clausura, entidades desconocidas o caracteres no reconocidos, de lo contrario se detiene el proceso de análisis, ya sea para extraer los elementos o para convertir en conjunto con las hojas de estilo.

Es así como se decide manipular el HTML por medio de las librerías que proporcionaba Tidy, la herramienta elegida para este caso:

Los métodos que se encuentran en la clase Tidy se manejaron con los valores estipulados a en el Listado 1 del Apéndice D, con el fin de lograr que el HTML recibido se estructurara como XML.

4.3.1 Caso: contenido de entidades desconocidas

En algunos casos se observó que a pesar de manejar el Unicode LATIN1, aún así el analizador reportaba entidades no reconocidas, como las “&acute” que representan acentos en código HTML, (ver Listado 4.4) la solución fue definir las entidades posibles que el analizador no reconocería dentro de un DTD (ver Apéndice C). Esto es porque los caracteres de los documentos XML siguen el estándar Unicode, el cual es una extensión del conjunto de caracteres ASCII. Unicode soporta todos los caracteres de los idiomas hablados, así como todos los símbolos matemáticos, entre otros. Soporta el inglés, japonés, chino, etc. por lo que en el DTD se definen entidades como lo son: ó, È, î, ü, y otras

entidades propias de las lenguas europeas (español, francés, italiano), pero no japonés ni lenguas de otro tipo.

Los documentos que utilizan una codificación diferente de UTF-8 o UTF-16 (las estándares en cuanto a páginas *web*) deben comenzar con una declaración XML que lo indique para que el intérprete sepa que entidades debe reconocer:

<?XML version=\"1.0\" encoding=\"ISO-8859-1\"?>

pero en el caso de páginas que utilicen otra codificación diferente a la establecida arriba, no se podrá hacer la conversión, a menos que se extienda el DTD establecido para esta aplicación y se tenga la equivalencia de cada carácter en idioma fuente al idioma de salida, en este caso, restringido por el Unicode ISO-8859-1.

```
<table nosave="" cellpadding="0">
    <tbody>
     <tr nosave="">
       <td><b>Alfredo S&aacute;nchez</b> <br>
     <small><i>Coordinador</i> del <a
 href="http://ict.udlap.mx/projects/u-dl-a.html">Programa de Bibliotecas Digitales</a>
    </small><br>
     <small><a href="http://biblio.udlap.mx"></a><i>Profesor Titular</i>
de<i> </i><a href="http://www.udlap.mx/%7Esistemas">Ingenier&iacute;a en Sistemas
Computacionales</a>      <br>
       </small><small><i>Coordinador</i>, Laboratorio de Tecnolog&iacute;as
Interactivas y Cooperativas (<a href="http://ict.udlap.mx">ICT</a>)</small><br>
     <small><i>Investigador </i>del Centro de Investigaci&oacute;n en Tecnolog&iacute;as
 de Informaci&oacute;n y Automatizaci&oacute;n (<a
 href="http://www.udlap.mx/%7Ecentia">CENTIA</a>) <br>
```

Listado 4.4 Caso de entidades desconocidas (extracto de código HTML del URL-

http://www.udlap.mx/~alfredo)

Una vez explicados los conceptos anteriores, se entiende por qué se maneja un DTD declarando las entidades que pueden provocar la interrupción del proceso al no ser reconocidas. Aún así, se puede generar un error de este tipo, porque hay una enorme cantidad de entidades que pudiera no reconocer, notificando de esto al usuario.

La consecuente salida del Tidy genera un archivo XML que es el que se maneja en el análisis con Sax y Saxon. A este archivo se le agregan las etiquetas que se explicaron en la Sección 4.2:

<?XML version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE editmos SYSTEM "editmos.dtd">

Teniendo como documento de salida un archivo XML con sus respectivas etiquetas como se aprecia en la Figura 4.3.

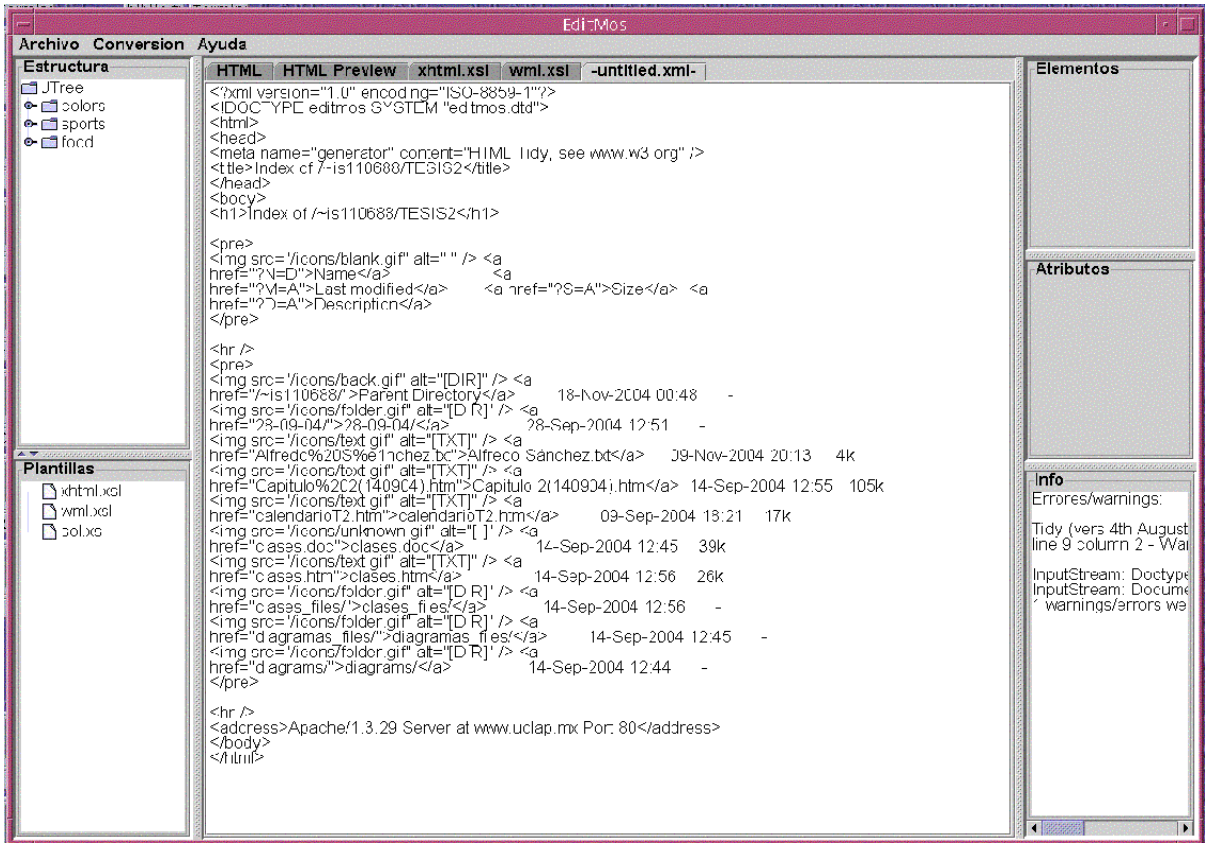

Figura 4.3 Salida del HTML Tidy

4.4 Procesamiento del archivo XML

Una vez obtenido el archivo XML a analizar, se inicia el análisis con SAX. SAX fue diseñado como un API ligero que no involucra la generación de estructuras internas. Las aplicaciones deben registrar manejadores de eventos a un objeto de análisis que implemente la interfaz org.sax.Analizador. SAX tiene tres interfaces para manejo de eventos (*handlers)*: DocumentHandler, DTDHandler, ErrorHandler. También implementa la clase *default* HandlerBase para el comportamiento *default* de estas interfaces.

DocumentHandler es la más importante y la más usada ya que es llamada cada vez que se encuentra un elemento (Listado 2 del Apéndice D).

El resultado del análisis, como ya se mencionó, es una lista de elementos con sus respectivos atributos, así como un árbol describiendo la estructura del archivo, como se observa en la Figura 4.4 por medio de las flechas.

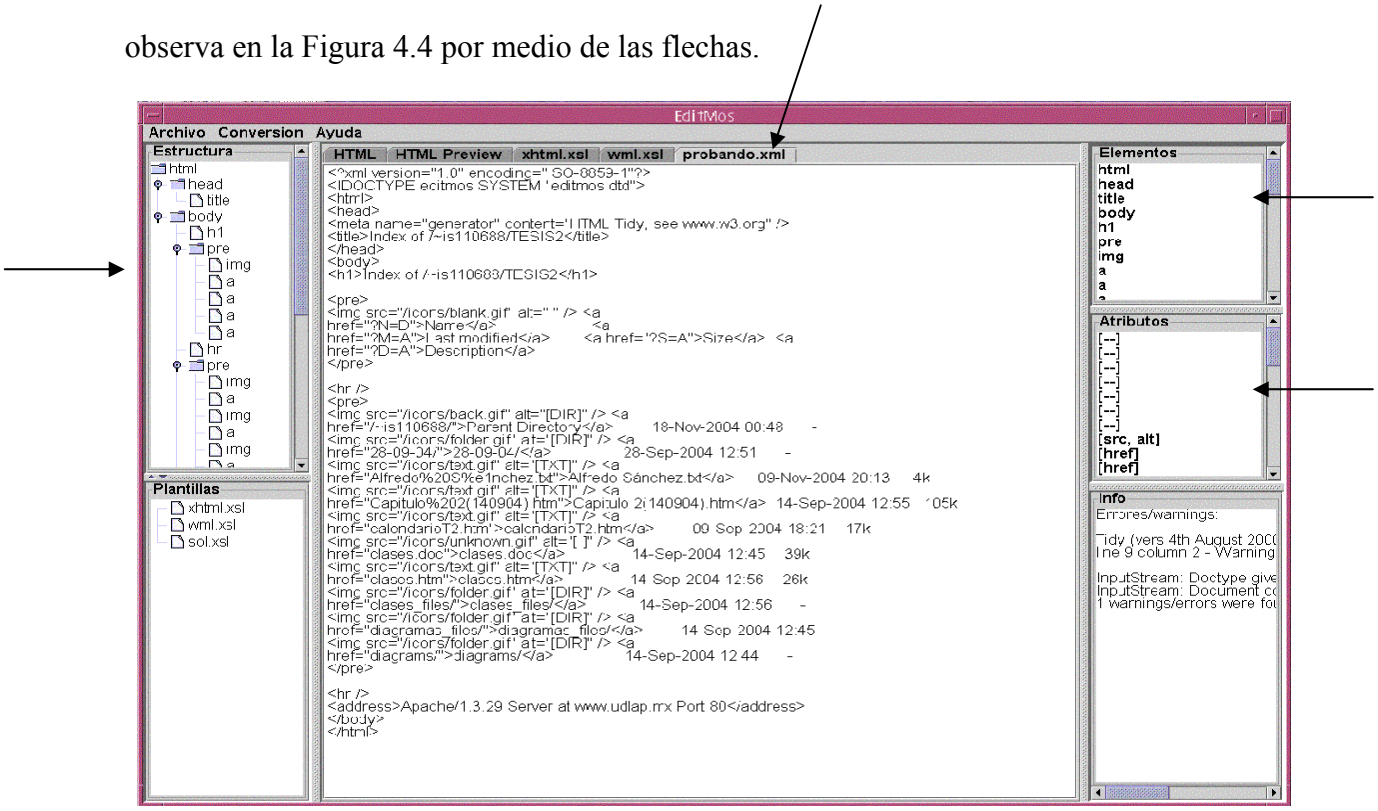

Figura 4.4 Análisis del archivo XML

4.5 Conversión a formatos de salida

El último paso del proceso de conversión es el que corresponde al analizador Saxon que como su nombre lo dice, implementa a SAX, el analizador elegido en conjunto a hojas de estilo XSL.

Saxon esta basado en SAX, por lo que se consideró como la mejor opción para seguir la línea de la tecnología implementada hasta este punto en el proyecto.

Saxon funciona con base a la comparación con las plantillas especificadas en el Apéndice E que contienen las equivalencias de las etiquetas en XML con las etiquetas en WML o XHTML. Toma la primera etiqueta, va a la plantilla (hoja de estilo), y si son iguales, ejecuta la acción especificada en cada caso, de no encontrar una equivalencia dentro de la plantilla, no podrá procesar ese elemento indicándoselo al usuario. El resultado es el archivo HTML equivalente en lenguaje WML o XHTML (ver Figura 4.5).

El archivo resultante se aprecia entonces en un Emulador Openwave SDK 6.2.2 para teléfonos móviles en el caso de WML, ilustrado en la Figura 4.6, y para PDA en el caso de XHTML, siendo la primera imagen, la vista original en un navegador, como se muestra en la Figura 4.7.

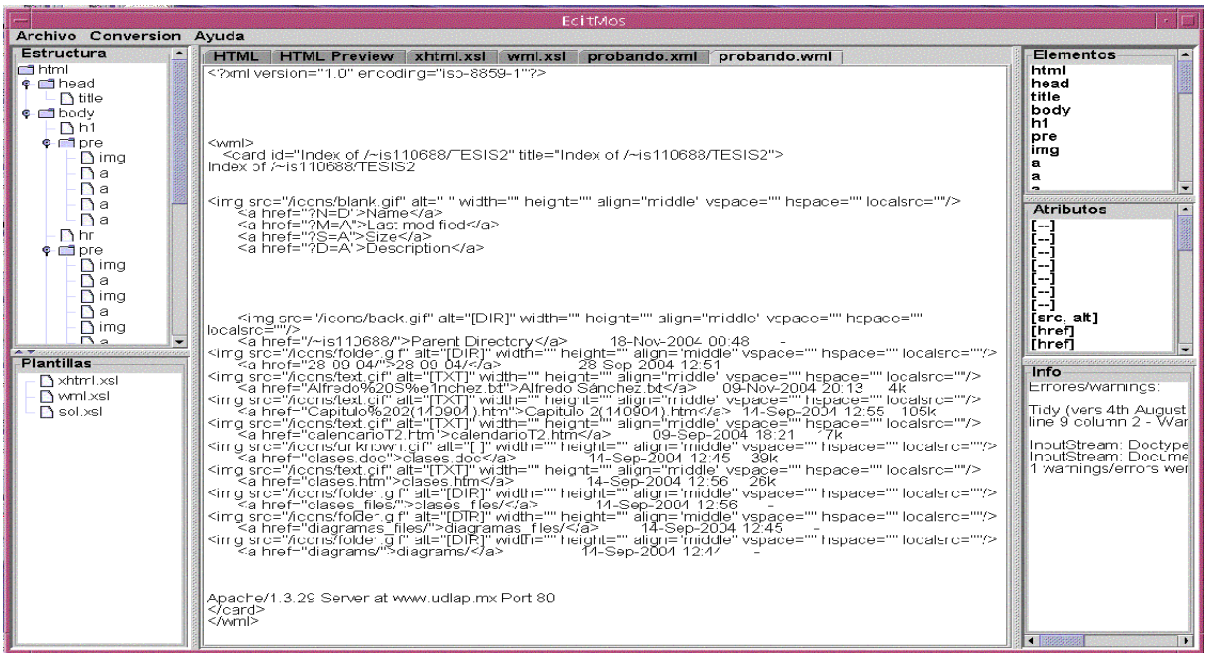

Figura 4.5 Salida del Saxon: archivo WML

Figura 4.6 Vista en emulador Openwave SDK 6.2.2

Una vez explicadas las clases implementadas para lograr los objetivos establecidos y mencionados dentro de este documento, se procede a las pruebas de la aplicación y los resultados obtenidos para posteriormente concluir con los avances obtenidos.

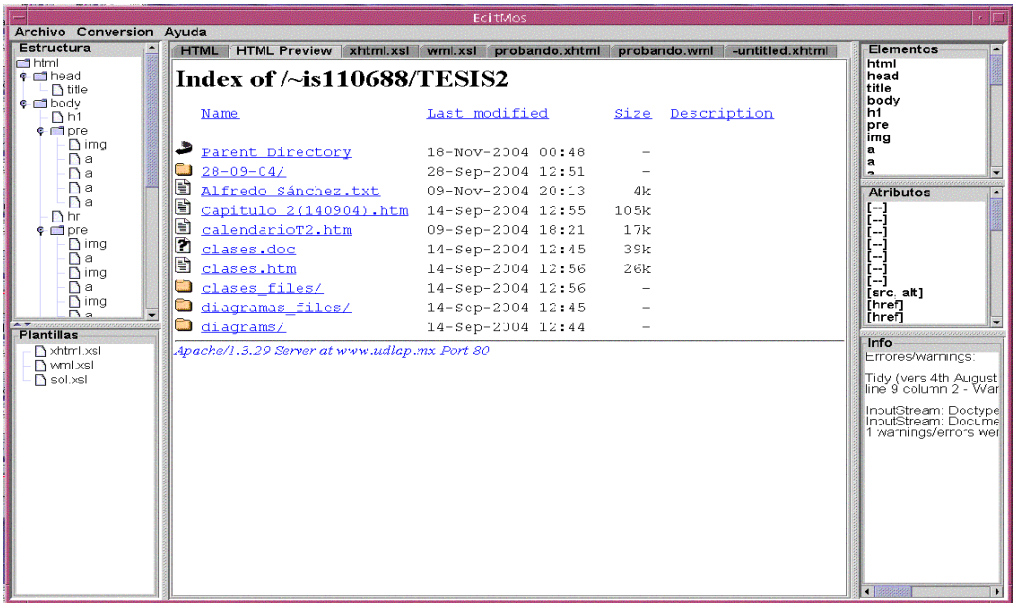

Figura 4.7 Vista en navegador