6. PROTOCOLO WEBDAV

6.1 Principales conceptos

WebDAV es un grupo de trabajo del IETF. El término significa "Edición y versionado distribuidos sobre la web", y se refiere al protocolo (más precisamente, la extensión al protocolo) que el grupo definió.

El objetivo de WebDAV es hacer de la World Wide Web un medio legible *y* editable. Este protocolo proporciona funcionalidades para crear, cambiar y mover documentos en un servidor remoto (típicamente un servidor web). Esto se utiliza sobre todo para permitir la edición de los documentos que contiene un servidor web, pero puede también aplicarse a sistemas de almacenamiento generales basados en web, que pueden ser accedidos desde cualquier lugar. La mayoría de los sistemas operativos modernos proporcionan soporte para WebDAV, haciendo que los ficheros de un servidor WebDAV aparezcan como almacenados en un directorio local.

WebDAV comenzó su andadura cuando Jim Whitehead propuso resolver el problema de la edición distribuida en la Web para discutir posibles soluciones. La visión original de la World Wide Web tal y como fue expuesta por Tim Berners-Lee era crear un medio donde cualquiera pudiera leer y editar. De hecho, el primer navegador creado por Tim, llamado WorldWideWeb, era capaz de visualizar y editar páginas remotas. Sin embargo, la web se desarrolló como un medio de solo lectura. El grupo de personas que formaban parte del W3C decidieron que el mejor camino para proceder era formar un grupo de trabajo del IETF. El IETF pareció una elección natural, debido a que el protocolo HTTP estaba siendo estandarizado allí y se asumía que la salida de este esfuerzo consistiría en extender este protocolo.

Cuando empezó el trabajo sobre el protocolo WebDAV quedó claro que para manejar la edición y el control de versiones distribuido era lógico separar estas dos tareas. El grupo de trabajo WebDAV decidió así concentrarse en la edición distribuida y dejar las versiones para el futuro.

Algunas de las características más destacadas de WebDAV (aparte de proporcionar acceso a la escritura del documento via http) son el locking o posibilidad de cerrar dicho acceso de escritura, mecanismo imprescindible en un entorno de trabajo compartido, para evitar que el trabajo de un nuevo usuario sobreescriba al del anterior. Además, ofrece la posibilidad de describir propiedades (metadatos) del documento en formato xml, como titulo, asunto, autor, fecha, tamaño, que aunque no aparecen en el documento proporcionan información sobre el mismo y pueden ser fácilmente gestionados por el protocolo DAV (permitiendo funciones de listado de datos, búsqueda inteligente, etc).

WebDAV también contempla la posibilidad de renombrar, mover, copiar o borrar documentos en el propio servidor y la posibilidad de llevar un registro de las sucesivas versiones o revisiones del documento (pudiendo recuperar cualquiera de las anteriores). También permitirá a los usuarios crear enlaces virtuales entre recursos.

Uno de los aspectos mas interesantes de WebDAV es la posibilidad de trabajar o escribir no sólo con documentos, textos. La posibilidad de trabajo conjunto se extiende a elementos HTML, GIF, JPEG, etc. De hecho ya existen aplicaciones gráficas que admiten este protocolo. WebDAV no pone

ninguna restricción al tipo de documentos (elementos) sobre los que se puede trabajar.

WebDAV permitirá a multitud de dispositivos escribir de forma segura en la red.

6.2 Métodos utilizados

WebDAV añade los siguientes métodos a HTTP:

- PROPFIND - Usado para recuperar propiedades, almacenadas como XML, desde un recurso. También está sobrecargado para permitir recuperar la estructura de colección (alias jerarquía de directorios) de un sistema remoto.
- PROPPATCH - Usado para cambiar y borrar múltiples propiedades de un recurso en una simple operación atómica (atomic commit).
- MKCOL - Usado para crear colecciones o directorios.
- COPY - Usado para copiar un recurso desde un URI a otro.
- MOVE - Usado para mover un recurso desde un URI a otro.
- LOCK - Usado para bloquear un recurso. WebDAV soporta tanto bloqueos compartidos como exclusivos.
- UNLOCK - Para desbloquear un recurso.

A continuación, pasaremos a analizar cada uno de los métodos empleados para poder gestionar documentos en un servidor WebDAV.

6.2.1 PROPFIND

El método PROPFIND recupera las propiedades definidas en el recurso definido por la URI solicitada, en caso de que el recurso no tenga dependencias, o las propiedades definidas también en el recurso solicitado y en sus dependencias, en caso de que se trate de una colección o directorio.

El cliente debe poseer un elemento denominado “Depth” con valor “0” ,“1” ó infinito, mientras que los servidores WebDAV deben soportar a su vez dichos valores. Por defecto, si no se especifica ningún valor en el campo Depth, actúa como si se incluyera la cabecera "Depth: infinity".

Un cliente puede poseer un elemento propfind XML en el cuerpo de la petición en el que se describe qué información se pide. Es posible requerir valores de propiedades concretas, o bien de todas las propiedades. En caso de que la petición propfind vaya vacía en el cuerpo del mensaje, se trataría como si se pidieran todas los nombres y valores de las propiedades.

Todos los servidores deben devolver una respuesta del tipo text/xml o application/xml que contenga

un elemento xml que describa los resultados requeridos en las peticiones. Si hay algún error en alguna de las propiedades, se debe incluir el error correspondiente como resultado en la respuesta. Una petición que contenga el valor de una propiedad que no exista, es un error y debe ser notificado en la respuesta conteniendo el elemento 404 (Not Found) en el valor del estado.

Como consecuencia, el elemento multi-estado XML para una colección de recursos, o lo que es lo mismo, para un directorio y sus elementos hijos, debe incluir una respuesta XML para cada miembro de la colección, sea cual sea el campo DEPTH especificado. Cada respuesta XML debe contener un elemento XML *href* que proporcione la URI del recurso cuyas propiedades son definidas. Los resultados a esta petición PROPFIND de una colección y sus dependencias son devueltas como una lista cuyo orden en las entradas no es significativo.

A continuación, se mostrarán distintos ejemplos de las peticiones/respuestas que se realizan cuando se invoca el método PROPFIND.

- Petición-Request:

```
PROPFIND /public/docs/myFile.doc HTTP/1.1
Content-Type: text/xml
Content-Length: XXX
Depth: 0
Translate: f
...

<?xml version="1.0"?>
<a:propfind xmlns:a="DAV:">
<a:prop><a:getcontenttype/></a:prop>
<a:prop><a:getcontentlength/></a:prop>
</a:propfind>
```

- Respuesta-Response:

```
HTTP/1.1 207 Multi-Status
Content-Type: text/xml
Content-Length: 310

<?xml version="1.0"?>
<a:multistatus
  xmlns:b="urn:uuid:c2f41010-65b3-11d1-a29f-00aa00c14882/"
  xmlns:a="DAV:">
 <a:response>
   <a:href>http://server/public/test2/item1.txt</a:href>
   <a:propstat>
    <a:status>HTTP/1.1 200 OK</a:status>
       <a:prop>
        <a:getcontenttype>text/plain</a:getcontenttype>
        <a:getcontentlength b:dt="int">33</a:getcontentlength>
       </a:prop>
   </a:propstat>
 </a:response>
</a:multistatus>
```

El ejemplo siguiente utiliza el elemento "*allprop*" para devolver todos las propiedades de una

colección. El campo DEPTH está puesto a 0, por lo que las propiedades se devuelven exclusivamente a la colección y no a los hijos de ésta.

- Petición-Request:

```
PROPFIND /public/container/ HTTP/1.1
Host: www.contoso.com
Depth: 0
Content-type: text/xml;
Content-Length: xxxx

<?xml version="1.0" ?>
<D:propfind xmlns:D="DAV:">
        <D:allprop/>
</D:propfind>
```

- Respuesta-Response:

```
HTTP/1.1 207 Multi-Status
Content-Type: text/xml
Content-Length: xxxx

<?xml version="1.0" ?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
       <D:href>http://www.contoso.com/public/container/</D:href>
       <D:propstat>
            <D:prop xmlns:R="http://www.contoso.com/schema/">
                 <R:author>Rob Caron</R:author>
                 <R:editor>Jessup Meng</R:editor>
                 <D:creationdate>
                    1999-11-01T17:42:21-06:30
                 </D:creationdate>
                 <D:displayname>
                    Example Collection
                 </D:displayname>
                 <D:resourcetype><D:collection></D:resourcetype>
                 <D:supportedlock>
                    <D:lockentry>
                       <D:lockscope><D:shared/></D:lockscope>
                       <D:locktype><D:write/></D:locktype>
                    </D:lockentry>
                 </D:supportedlock>
              </D:prop>
              <D:status>HTTP/1.1 200 OK</D:status>
          </D:propstat>
        </D:response>
</D:multistatus>
```

Como se ha comentado anteriormente, existen distintos estados que pueden ser devueltos en una respuesta. De esta forma, se conoce como respuesta multi-estado. Los distintos valores que podemos encontrar son los siguientes:

1. 200 (OK). Respuesta correcta.

2. 403 (Forbidden/Prohibido). El cliente no tiene permisos para acceder al host requerido.

3. 404 (Not Found/No encontrado). Las propiedades especificadas no han sido encontradas.

4. 409 (Conflict/Conflicto). El cliente ha especificado unas propiedades cuya semántica no es apropiada.

Un ejemplo de la implementación de dicho método en lenguaje java es el siguiente:

```
public class PropFindMethod extends AbstractDAVMethod
{
    private Depth depth;
    private PropFindQueryType queryType;
    private DAVProperty[] queryProps;

    public PropFindMethod(String url)
    {
      super(url);
        this.depth=Depth.ONE;
        this.queryType=PropFindQueryType.ALLPROP;
    }
}
```

Una posible llamada a este método sería:

```
 PropFindMethod method;
                if (DO_ALLPROP)
                  method=new PropFindMethod(initialPath);
                else
                  method=new PropFindMethod(initialPath, Depth.ONE,
DAVProperty.BASIC_PROPERTIES);

                  dc.execute(method);
```

6.2.2 PROPPATCH

El método PROPPATCH procesa las instrucciones especificadas en el cuerpo de la petición para añadir y/o quitar las propiedades definidas en el recurso identificado por la URI en la petición.

Todos los recursos DAV deben soportar el método PROPPATCH y deben procesar las instrucciones que se especifican usando los elementos xml *propertyupdate*, *set* y *remove* del esquema DAV.

El cuerpo del mensaje de la petición del método PROPPATCH debe contener el elemento xml *propertyupdate*, tal y como se comentó anteriormente. Las instrucciones deben ser procesadas en el orden en que fueron recibidas. Así mismo, se deben ejecutar todas o ninguna. Así, si ocurre algún error durante el proceso de ejecución de las instrucciones, deben ser deshechas todas las órdenes ejecutadas hasta el momento y devolver el error correspondiente.

A continuación se muestran los correspondientes mensajes XML del método PROPPATCH:

- Petición-Request:

```
PROPPATCH /bar.html HTTP/1.1
 Host: www.foo.com
 Content-Type: text/xml; charset="utf-8"
 Content-Length: xxxx

 <?xml version="1.0" encoding="utf-8" ?>
 <D:propertyupdate xmlns:D="DAV:"
 xmlns:Z="http://www.w3.com/standards/z39.50/">
   <D:set>
        <D:prop>
             <Z:authors>
                  <Z:Author>Jim Whitehead</Z:Author>
                  <Z:Author>Roy Fielding</Z:Author>
             </Z:authors>
        </D:prop>
   </D:set>
   <D:remove>
        <D:prop><Z:Copyright-Owner/></D:prop>
   </D:remove>
 </D:propertyupdate>
```

- Respuesta-Response:

```
HTTP/1.1 207 Multi-Status
 Content-Type: text/xml; charset="utf-8"
 Content-Length: xxxx

 <?xml version="1.0" encoding="utf-8" ?>
 <D:multistatus xmlns:D="DAV:"
 xmlns:Z="http://www.w3.com/standards/z39.50">
   <D:response>
        <D:href>http://www.foo.com/bar.html</D:href>
        <D:propstat>
             <D:prop><Z:Authors/></D:prop>
             <D:status>HTTP/1.1 424 Failed Dependency</D:status>
        </D:propstat>
        <D:propstat>
             <D:prop><Z:Copyright-Owner/></D:prop>
             <D:status>HTTP/1.1 409 Conflict</D:status>
        </D:propstat>
        <D:responsedescription> Copyright Owner can not be deleted or
 altered.</D:responsedescription>
   </D:response>
 </D:multistatus>
```

En este ejemplo, el cliente solicita al servidor establecer el valor http://www.w3.com/standards/z39.50/Authors a la propiedad correspondiente, y eliminar la propiedad http://www.w3.com/standards/z39.50/Copyright-Owner. La respuesta devuelve el error 425 (Failed), puesto que la propiedad pedida no puede ser eliminada. Además, también devuelve el error 409 (Conflict) que indica que no se ha podido realizar la acción porque ha fallado una anterior.

6.2.3 GET

El método GET permite obtener toda la información correspondiente de una colección y/o sus hijos

correspondientes. Cuando se aplica en concreto a una colección, puede devolver los contenidos de un recurso del tipo "index.html", una vista en la que se puede leer los contenidos de la colección.

La implementación Java en código del método sería la siguiente:

```
public class GetMethod extends AbstractDAVMethod
{

    public GetMethod(String url)
    {
        super(url);
    }

    public final DAVMethodName getRequestMethodName()
    {
        return DAVMethodName.GET;
    }
}
```

Y una posible llamada a este método:

```
 public byte[] get(DAVFile file)
    {

      GetMethod method=new GetMethod(file.getName());
      byte[] body=null;
      try
      {
          dc.execute(method);
          int status=method.getStatus();
          body=method.getResponseBody();
      }
}
```

El método GetMethod se basa en la librería commons-httpclient.

6.2.4 PUT

El método PUT puede ser aplicado tanto a recursos que son colecciones como a los que no lo son. En caso de no ser una colección, el método PUT es el responsable de actualizar las propiedades definidas en la petición que corresponden a un determinado recurso. Por ejemplo, si un servidor reconoce el tipo de contenido del cuerpo de la petición, debe ser capaz de extraer automáticamente la información que puede ser definida como propiedades.

Si el método PUT falla en la creación del recurso con esas nuevas propiedades, la respuesta debe devolver el error correspondiente: 409 (Conflict).

En caso de que el método PUT sea aplicado a una colección, la petición XML implicará la creación y/o borrado de recursos. La especificación correspondiente [RFC 2068] no especifica el formato de transmisión para crear una colección nueva con el método PUT. En su lugar, especifica el método MKCOL para crear colecciones.

Cuando se crea un nuevo recurso en una colección, ésta y todos sus antecesores especificados deben existir; es decir, si se desea crear /a/b/c/d.html, /a/b/c/ debe existir previamente. En caso de que no sea así, se deberá devolver el error 409 (Conflict).

Una posible implementación de dicho código en lenguaje Java sería la siguiente:

```
public void put(final DAVTreeNode parentNode,
                final String fileName,
                final byte[] bodyBytes)
   {
     put(parentNode, fileName, bodyBytes, null);
   }
```

6.2.5 Propiedades DAV

En cuanto a las propiedades DAV, el nombre de la propiedad en cuestión es también el mismo que el del elemento XML que contiene su valor. A continuación veremos las propiedades más significativas y las que se han usado en el desarrollo de la aplicación de este proyecto.

1. Displayname

   Proporciona un nombre adecuado al recurso que representa. La propiedad Displayname debe ser definida en todos los recursos DAV.

2. GetContentLanguage

   Contiene la cabecera “Content-Language” devuelta por el método GET.

3. GetContentLength

   Contiene la cabecera “Content-Length” devuelta por el método GET.

4. GetContentType

   Contiene la cabecera Content-Type devuelta por el método GET.

5. GetLastModified

   Contiene la cabecera Last-Modified devuelta por el método GET. La fecha de la última modificación del recurso debe reflejar los cambios o modificaciones de cualquier parte del recurso.

6.2.6 Privilegios

Para poder realizar distintas operaciones sobre un recurso, existen distintos privilegios o permisos. A continuación se analizará los distintos privilegios existentes.

1. Lectura

   El permiso de lectura controla los métodos que devuelven información sobre el estado del recurso, incluyendo sus propiedades. Los métodos afectados son GET y PROPFIND.

2. Escritura

   El permiso de escritura controla los métodos que bloquean o modifican un recurso, ya sea su contenido, propiedades, o en caso de una colección, alguno de sus hijos. Los métodos que realizan estas opciones son PUT y PROPPATCH.

3. Escritura propiedades

   El permiso write-properties o permiso de escritura de propiedades, controla los privilegios que modifican las propiedades de un recurso, tal y como su nombre indica. El método que realiza esta acción es PROPPATCH.

4. Escritura contenido

   El permiso de escritura del contenido controla los métodos que modifican el contenido de un recurso existente, tal y como hace el método PUT.

5. Bind

   El privilegio bind permite a los métodos añadir un nuevo miembro a la URL de la colección especificada. Dichos métodos son PUT y MKCOL. Dicho permiso se ignora en caso de que el recurso no sea una colección.

6. Unbind

   El permiso unbind permite a un método eliminar un miembro de la colección especificada en la URL. Dichos métodos son DELETE o MOVE. Al igual que el permiso anterior, se ignora en caso de que el recurso no sea una colección.

A continuación se muestra una tabla con los privilegios necesarios sobre los distintos recursos para poder ejecutar los métodos correspondientes:

Método	**Privilegios**
GET	Lectura
OPTIONS	Lectura
PUT	Escritura contenido
PROPPATCH	Escritura propiedades
PROPFIND	Lectura
COPY	Lectura, escritura contenido y escritura propiedades

MOVE	Bind y unbind
MKCOL	Bind
CHECKOUT	Escritura propiedades

Tabla 6.1. Permisos asociados a los distintos métodos

6.3 Ejemplo práctico de Servidor WebDAV

A continuación se mostrará un ejemplo práctico de un servidor WebDAV, así como la información que se puede extraer de los elementos contenidos.

La siguiente figura muestra un servidor WebDAV al que se puede acceder vía web HTTP. El servidor almacena directorios y documentos, más concretamente facturas y albaranes electrónicos que deben ser firmados:

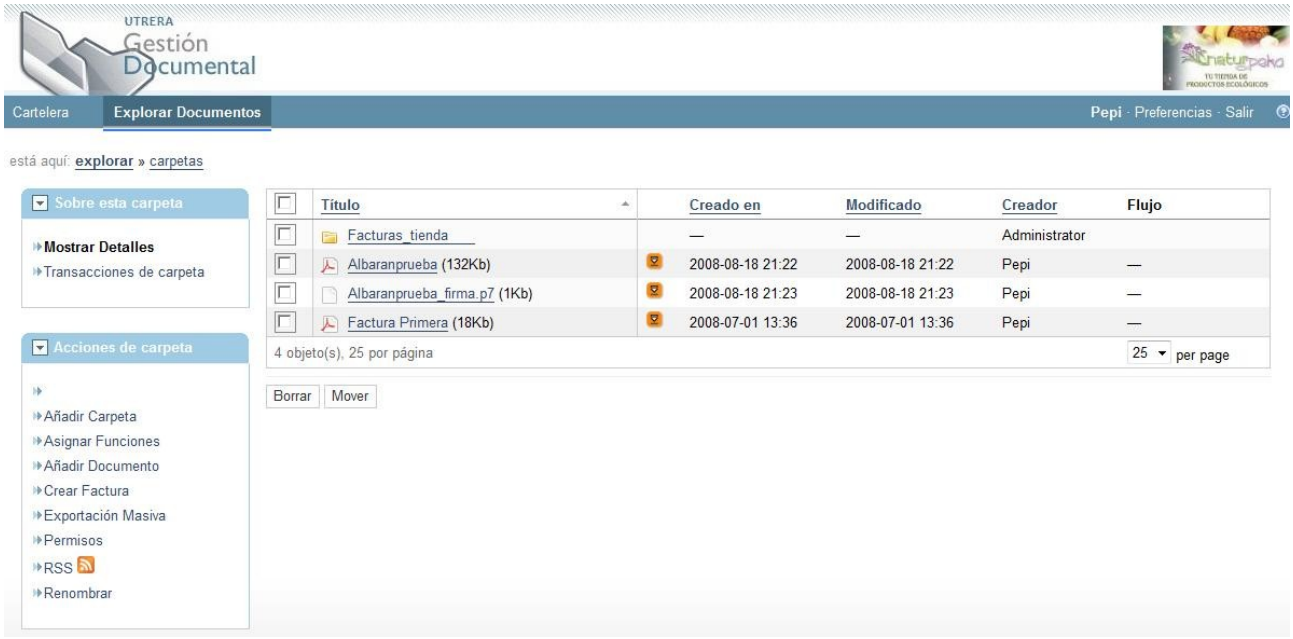

Figura 6.1. Ejemplo de servidor WebDAV

Con el siguiente código Java podemos obtener un documento existente en el servidor y todas las propiedades asociadas a él:

```
ResourceBundle prop = ResourceBundle.getBundle("resources.conexion");

        String host = prop.getString("conexion.host");
        String puerto=prop.getString("conexion.port");
        int port = Integer.parseInt(puerto);

        String filename="Factura Prueba.pdf";
```

```
DAVConnection dc=DAVConnectionPool.getDAVConnection(host, port, false);

      dc.setUsername("username");
      AuthorizationInfo ai= new AuthorizationInfo(host,
                  port,
                  "Basic",
                  "KTWebDav Server",
                  "password");
      dc.setAuthorization(ai.toString());


      GetMethod method = new GetMethod(filename);
      dc.execute(method);

      byte[] contenido=method.getResponseBody();

      String content = new String(contenido,"UTF8");

      PropFindMethod method2=new PropFindMethod(filename);
      dc.execute(method2);
      DAVFile archivo=method2.getDAVFile();
      String nameArchivo=archivo.getDisplayName();
      String Date=archivo.getLastModified();
```

Con el método GetMethod podemos obtener el contenido del documento solicitado, mientras que con PropFindMethod podemos obtener todas las propiedades asociadas al documento, como por ejemplo, la fecha de la última modificación, así como la ruta completa del archivo.

En el ejemplo anterior, host representa la dirección URL donde se encuentra el servidor WebDAV; por el puerto donde se encuentra alojado; username y password el nombre de usuario y contraseña para acceder al servidor WebDAV en caso de que los hubiera.

6.4 Clientes WebDAV

A continuación se realizará una breve enumeración del software disponible que soporta el protocolo WebDAV:

1. Jakarta Slide.

   Este proyecto fue uno de los principales proyectos que implementaban el protocolo WebDAV utilizando el lenguaje Java. El grupo de investigación retiró el proyecto en Noviembre de 2007. Una figura que ilustra la arquitectura sería la siguiente:

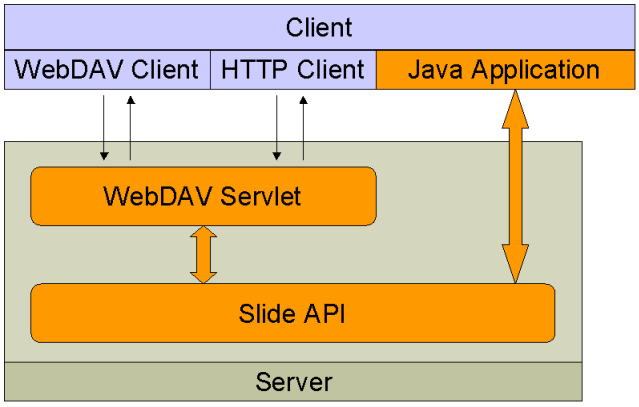

Figura 6.2. Arquitectura proyecto Jakarta Slide

2. Apache Tomcat. Tomcat posee un servlet WebDAV que puede ser usado para escribir y leer contenido desde una aplicación Web. Es de código libre, OpenSource.

3. DAV Explorer. Se trata de una herramienta GUI para explorar servidores DAV. Ha sido actualizado recientemente para poder ser usado con nuevas especificaciones, por lo que debería ser operable con la mayoría de los servidores.

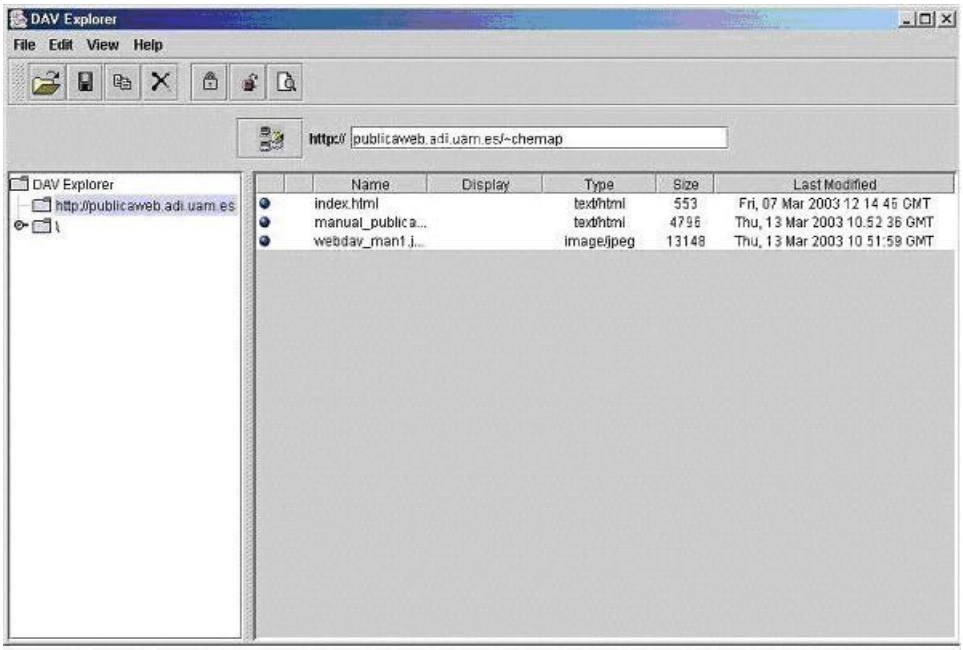

Figura 6.3. Entorno gráfico del cliente DAV Explorer

4. SkunkDAV. Es una librería Java que permite acceder a un servidor WebDAV. Es de código OpenSource, y permite acceder mediante una interfaz gráfica a los documentos, editarlos, crear nuevas carpetas, descargar o actualizar documentos,..Concretamente, ha sido el código base para crear el cliente Java que permitiera acceder a los documentos del servidor WebDAV de la aplicación.

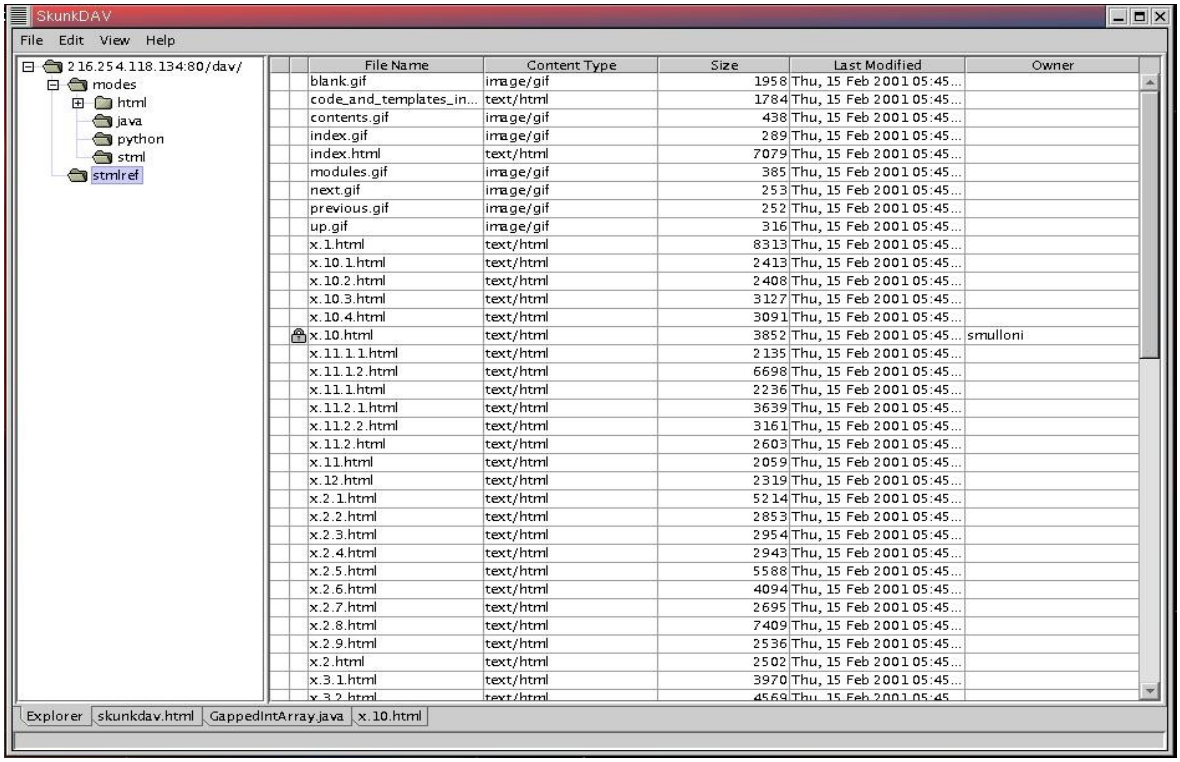

Figura 6.4. Entorno gráfico cliente SkunkDAV

Hasta aquí, hemos analizado las principales opciones que existen para implementar clientes que se comuniquen con servidores WebDAV. Todos ellos son OpenSource, es decir, de código abierto y disponible para todos los usuarios.