Capítulo 4

Implementación

Este capítulo presenta la implementación de la herramienta Voai. El objetivo de la implementación de Voai fue demostrar la viabilidad de la arquitectura propuesta en el capítulo anterior. El sistema fue implementado usando *Java Servlets*. Java ofrece un conjunto de mecanismos y métodos para desarrollar aplicaciones complejas que puedan ser adaptables, dinámicas, entre otras cosas. Además, una de las principales ventajas de usar Java es la capacidad de portabilidad entre distintas plataformas; en otras palabras, Java es un lenguaje multiplataforma.

Todos los *servlets* que componen a Voai son descritos por los diagramas de clase UML que se pueden consultar en el apéndice A de este documento. En el apéndice B se explica el proceso de instalación de Voai. En el apéndice C se proporciona un manual de usuario para que cualquier duda sobre el uso de Voai sea debidamente resuelta.

La sección 4.1 presenta la correspondencia entre los componentes del diseño conceptual y los servlets que componen a Voai en la implementación real. La sección 4.2 describe el proceso de generación de un servidor OAI en tres fases. Finalmente la sección 4.3 concluye el capítulo mencionando las principales características y ventajas del software desarrollado en este trabajo.

4.1 Implementación de componentes

Para la implementación de la herramienta Voai, se hizo un fiel seguimiento del diseño conceptual presentado en el capítulo anterior. Se conservaron todos y cada uno de los componentes, lo que permitió hacer una correspondencia entre el diseño conceptual y la implementación real para cada uno de los componentes de Voai.

4.1.1 Servlets de especificación

En el capítulo tres de esta tesis se mencionó que existen seis peticiones del protocolo OAI-PMH. En ese mismo capítulo se mencionó el nombre y la función de cada petición. Sin embargo, se les dieron nombres genéricos a las peticiones para que el diseño fuera más entendible. En este punto es importante mencionar el nombre real de las peticiones OAI-PMH para poder manejar formalmente los conceptos utilizados durante la explicación de este capítulo.

- El nombre real de la petición “identificación del servidor” es Identify
- El nombre real de la petición “manejo de conjuntos” es ListSets
- El nombre real de la petición “soporte de formatos de metadatos” es ListMetadataFormats
- El nombre real de la petición “recuperación de un registro” es GetRecord
- El nombre real de la petición “recuperación de metadatos de registros” es ListRecords
- El nombre real de la petición “recuperación de identificadores de registros” es ListIdentifiers

Una vez que se sabe el nombre real de las peticiones, se puede explicar la forma en que se mapeo el diseño a la implementación y el por qué de los nombres usados en esta última.

Los servlets de especificación son la implementación de lo que en el capítulo anterior se denominó módulos de especificación, es decir, a cada módulo de especificación le corresponde un servlet de especificación. Esta correspondencia la podemos ver claramente en la tabla 4.1.

Tabla 4.1. Correspondencia entre módulos y servlets.

Módulo de especificación	Servlet de especificación
Módulo para la identificación del servidor	Construye Identify
Módulo para el manejo de conjuntos	Construye ListSets
Módulo para el soporte de formatos de metadatos	Construye ListMetadataFormats
Módulo para la recuperación de un registro	Construye GetRecord
Módulo para la recuperación de varios registros	Construye ListRecords y ListIdentifiers

Cada uno de estos servlets recibe información (consultas SQL, datos para la conexión a la base de datos e información de identificación de la colección) y la almacena en el objeto Java de tipo Registro que le corresponde. Por ejemplo, cuando se suministra información al servlet Construye Identify, esta información se guarda en un objeto Java de tipo Registro Identify. La información del servlet Construye ListSets se almacena en el objeto Java de tipo Registro ListSets, y así análogamente para los demás servlets. Al final del proceso de

suministro de información, deben existir cinco objetos tipo Registro, uno por cada servlet de especificación.

La interacción con los servlets de Voai es muy flexible. Es posible que el administrador especifique información y posteriormente la modifique tantas veces como sea necesario sin que esto implique proporcionar nuevamente toda la información por cada modificación. Esto es posible gracias a que la información se encuentra en una tabla Hash que es recuperada y guardada en el sistema de archivos constantemente. Cuando se accede a un servlet, los campos de este son llenados a partir de la información previamente guardada en la tabla Hash. Cuando se actualiza la información del servlet, esta es almacenada nuevamente en la tabla Hash y se guarda otra vez en el sistema de archivos para ser utilizada en el proceso de generación de código, o para su posible actualización futura.

4.1.2 Tabla Hash

La tabla Hash es la manera en que se implementó el componente “base de especificaciones”. En la subsección 3.3.2 se había dicho que la información de cada módulo de especificación era guardada en una base de especificaciones. También se dijo que este proceso continuaba hasta que se guardaba toda la información de todos los módulos.

De forma análoga, la forma de guardar toda la información suministrada a los servlets de especificación es guardando los cinco objetos Registro en la tabla Hash. Esto se debe a que cada Registro tiene la información del servlet de especificación que le corresponde.

4.1.3 Servlet generador de código

Este servlet es la implementación del componente “generador de código” discutido en la subsección 3.3.3 del capítulo anterior. Este servlet recupera los cinco objetos Registro de la tabla Hash. Cuando ya se tienen estos objetos en memoria, se construyen las partes del código que responden a los verbos del protocolo OAI-PMH en base a la información contenida en esos objetos. Al final el código fuente que se generó (clases Java) es colocado en una carpeta temporal y permanece en ese lugar hasta que se necesite en la última fase de la generación del servidor OAI. Además, este servlet también genera un archivo instalador (Install.bat)

4.1.4 Carpeta temporal

Una simple carpeta contenida en Voai es denominada “carpeta temporal”. Esta carpeta, como su nombre lo dice, almacena temporalmente las clases Java generadas, y además contiene un archivo “descriptor de componentes”.

4.1.5 Descriptor de componentes

Web.xml o descriptor de componentes es un archivo que contiene la lista de los componentes del servidor OAI que será generado, es decir, contiene la lista de los servlets que reciben y procesan las seis peticiones del protocolo OAI-PMH. Este archivo esta almacenado en una carpeta de Voai y será una pieza clave para poder hacer la construcción de servidores OAI.

4.1.6 Archivo de instalación

En la sección 3.3.4 se mencionó que el generador de código era el responsable de crear al componente “Instalador”. Análogamente, el servlet Generador de Código es el responsable de crear un archivo de instalación (Install.bat) que corresponde a ese componente Instalador. Este archivo construye una estructura de archivos adecuada que se convertirá en el servidor OAI. Además, también se encarga de recuperar el código Java y el archivo Web.xml de una carpeta temporal, y los almacena en la estructura de archivos que él mismo creó. Después de la colocación adecuada del código Java y del archivo Web.xml en la estructura de archivos creada, el servidor esta listo para ser colocado en un contenedor de servlet de un servidor Web. Por ejemplo, se puede colocar en Tomcat [Tomcat 2005], y de esta manera ya puede estar listo para ser accedido por proveedores de servicios o por aplicaciones Web en general.

4.2 Generación de un servidor OAI en tres fases

Ahora que ya se conocen los nombres de los componentes en la implementación, a continuación se mencionará el proceso a seguir para generar un servidor OAI con Voai. Este proceso esta conformado por tres fases. Básicamente la fase uno consiste en brindar información a los servlets de especificación, la fase dos consiste en seleccionar la ubicación donde se desea que se instale el servidor OAI, finalmente la fase 3 consiste en generar e instalar el servidor OAI en la ruta previamente especificada. Estas tres fases se pueden ver claramente en la figura 4.1.

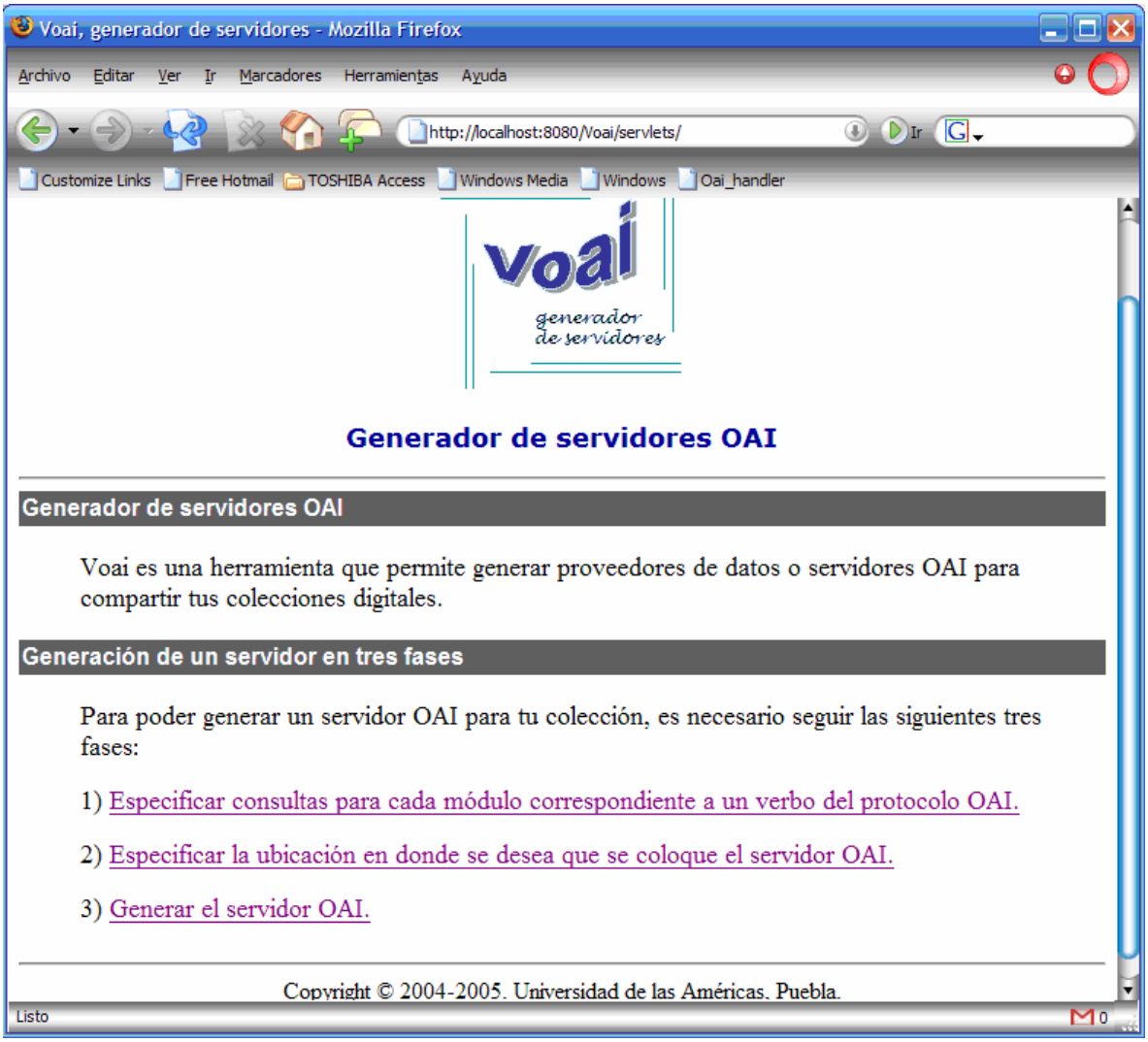

Figura 4.1. Interfaz principal de Voai

4.2.1 Fase uno: especificación de información

En esta primera fase básicamente lo que se hace es suministrar información a los servlets de especificación. Para esto se tiene un servlet principal con cinco hipervínculos que llaman a esos cinco servlets. Ese servlet se puede ver en la figura 4.2. Esta fase culmina cuando se haya proporcionado toda la información mínima necesaria a todos los servlets de especificación.

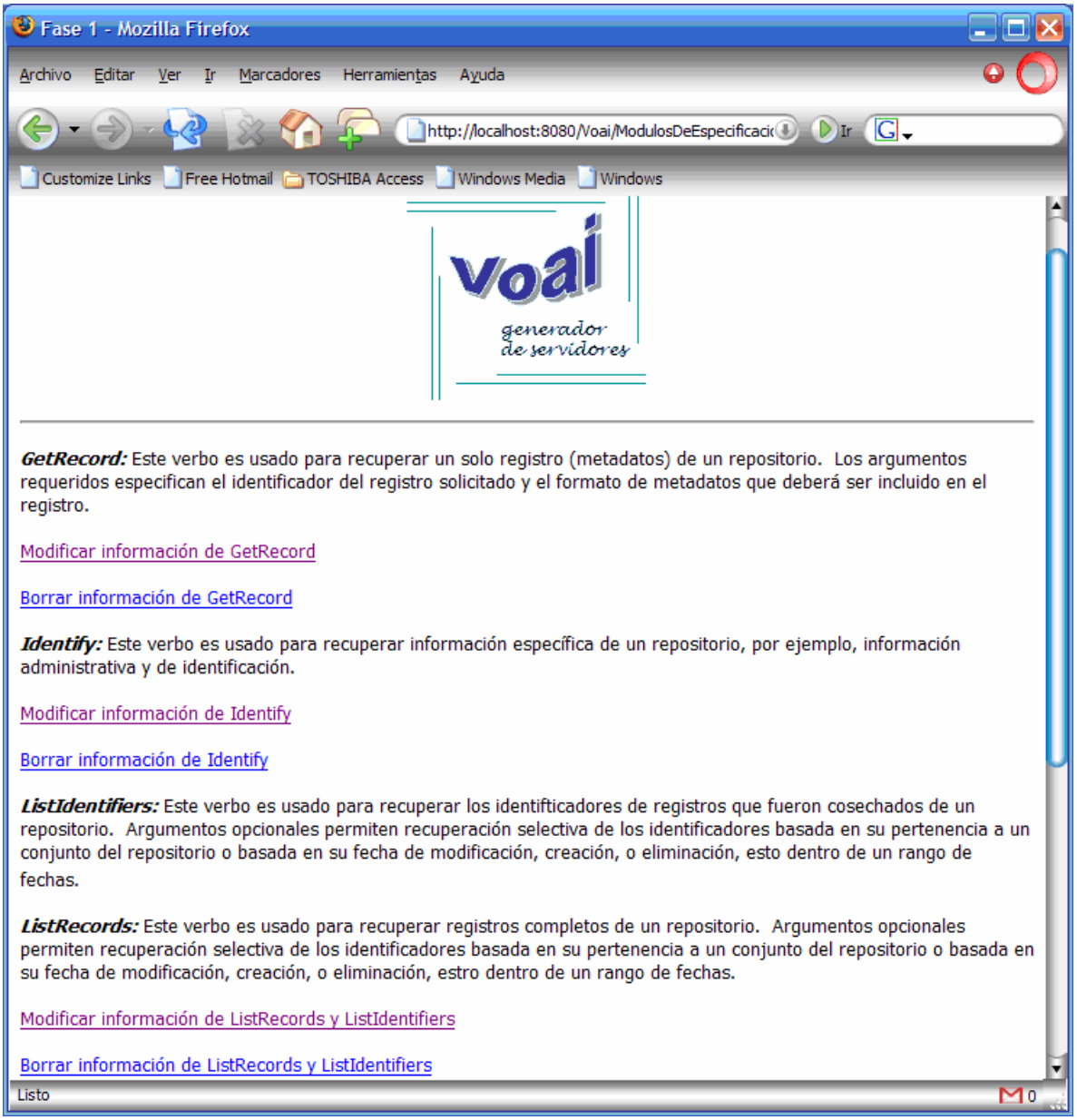

Figura 4.2. Fase 1 de Voai

En la figura 4.3 se muestra un ejemplo de un servlet de especificación, en este caso, es un servlet correspondiente al verbo Identify. Se puede ver que previamente ya se había proporcionado información de alguna colección real, y se tiene la posibilidad de modificar la información que se desee.

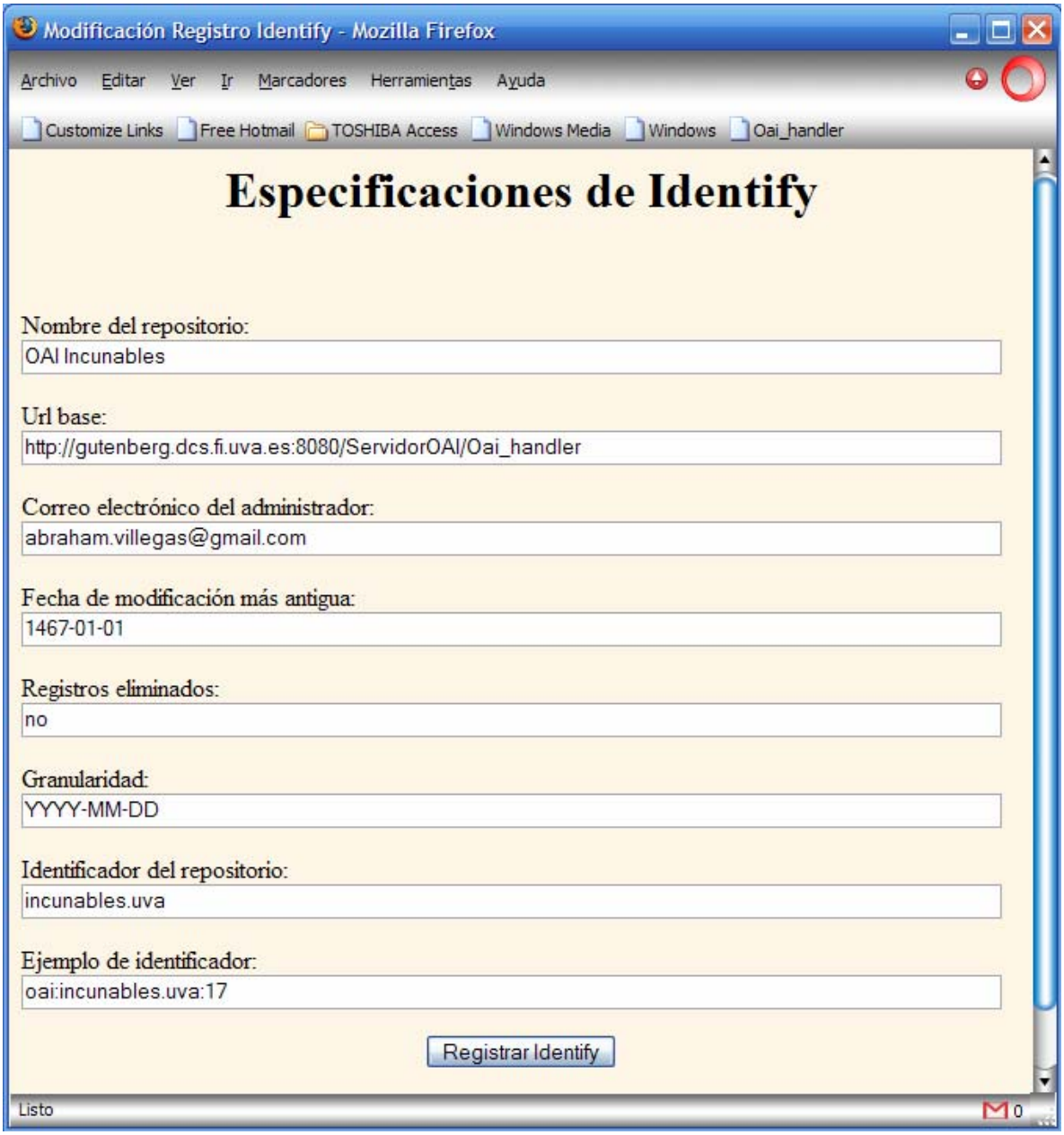

Figura 4.3. Servlet de especificación correspondiente a Identify

4.2.2 Fase dos: selección de ubicación de instalación

Esta fase consiste simplemente en seleccionar la ruta en donde se desea que sea instalado el servidor OAI. En caso de que ya se tenga un contenedor de servlets instalado, como Tomcat por ejemplo, se recomienda que la ruta seleccionada sea la de “webapps” de

Tomcat. De esta manera el usuario ya no tendrá que mover el servidor OAI al contenedor de servlets. Lo único que tendrá que hacer es ejecutar el servicio de Tomcat, y el servidor OAI podrá ser accedido a partir de ese momento. El servlet que representa esta fase es mostrado en la figura 4.4.

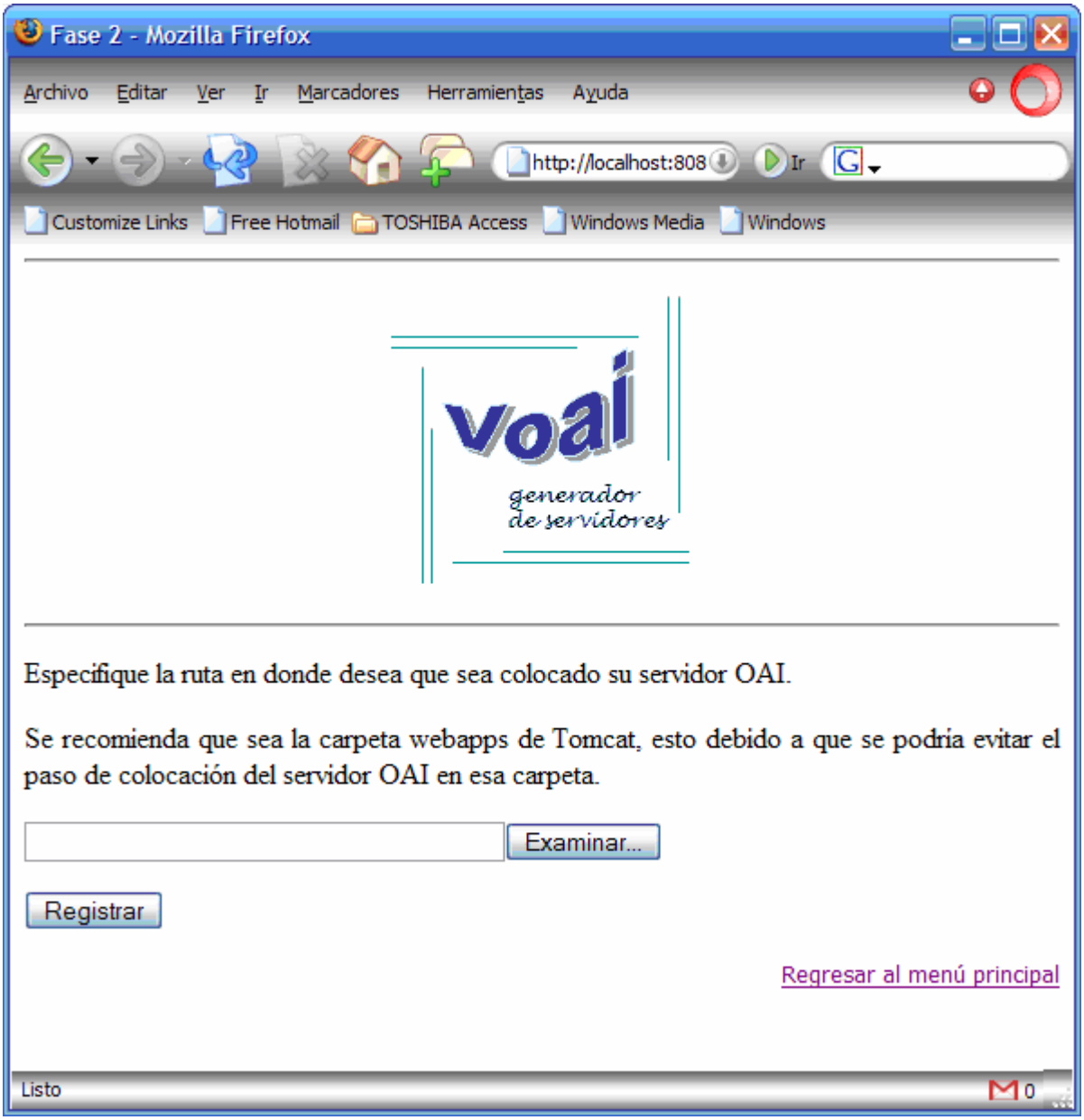

Figura 4.4. Fase 2 de Voai

4.2.3 Fase tres: generación del servidor OAI

En esta última fase se hace la generación del código del servidor y la generación del archivo de instalación. Cuando se selecciona el tercer hipervínculo de la interfaz principal (figura 4.1), se ejecuta el proceso de generación de código. Este proceso, como ya se ha mencionado previamente, consiste en obtener la información que fue proporcionada en los servlets de especificación y a partir de esa información se construye el código que implementa el protocolo OAI-PMH. Después de que el código Java terminó de ser generado, se crea un archivo instalador (Install.bat) en la ruta que fue especificada en la fase dos. Inmediatamente después aparece un servlet indicando que el servidor OAI ha sido generado y se solicita que se ejecute el archivo Install.bat para instalarlo por completo en la ubicación seleccionada. Este servlet se muestra en la figura 4.5.

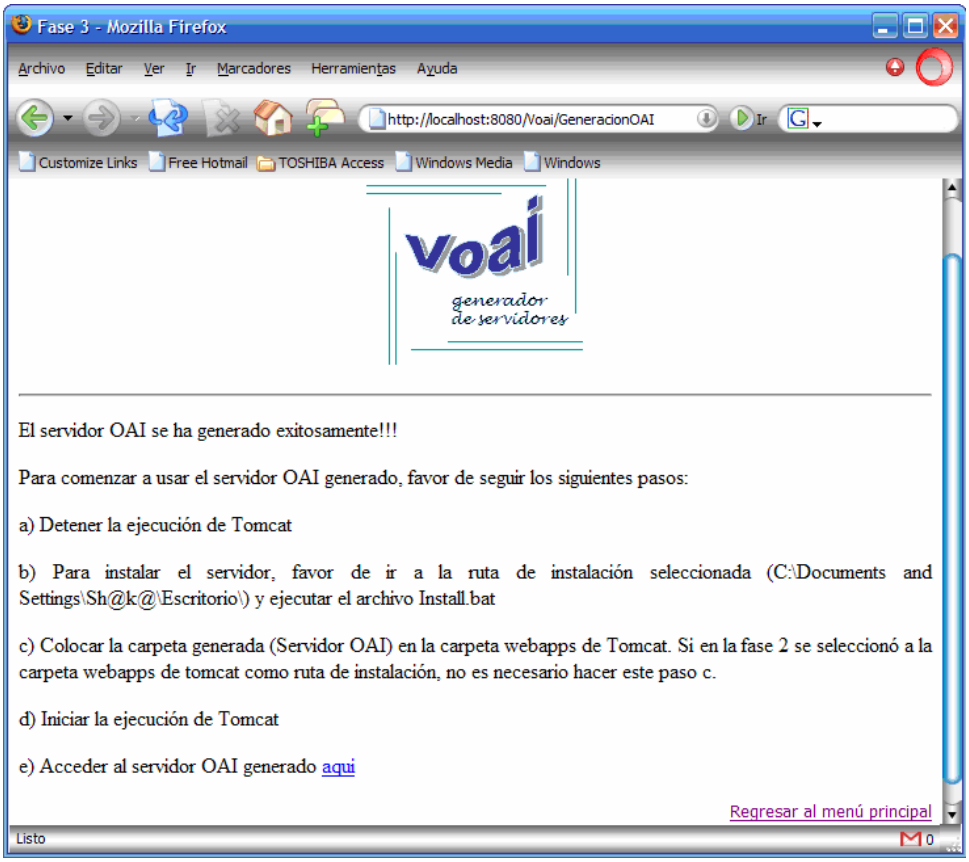

Figura 4.5. Fase 3 de Voai

4.3 Conclusiones

En este capítulo se presentó la implementación de la herramienta Voai. Esta implementación permite la generación de servidores OAI de manera automática. Para esto, solamente es necesario que el usuario administrador proporcione información sobre cómo recuperar metadatos de la colección, cuáles son los parámetros para conectarse a la base de datos, y cuál es la información de identificación de la colección.

Como se pudo ver, el usuario requiere conocer a fondo la colección que desea compartir. Además, debe conocer el lenguaje de consulta SQL y tener nociones del protocolo OAI-PMH. El proceso para generar un servidor OAI es sencillo y se traduce a la ejecución de tres fases. Voai facilita la construcción de servidores OAI a partir de colecciones digitales implementadas bajo un RDBMS, y para esto no es necesario que el usuario administrador de bases de datos realice algún tipo de implementación.