



Lavinia:

Ambiente WEB para PLN

Manual de Usuario

Version 1.2
Noviembre 2007

UNIVERSIDAD DE LA REPÚBLICA
Facultad de Ingeniería
Instituto de Computación
Montevideo, Uruguay

Índice

1. Introducción	4
2. Estructura de Lavinia	5
3. Instrucciones de Instalación	7
3.1. JBoss	7
3.2. Tomcat	8
3.3. Configuración	8
4. Funcionalidades	10
4.1. Componentes	10
4.1.1. Agregar Componente	10
4.1.2. Agregar Flujo	14
4.1.3. Ver	15
4.1.4. Modificar	15
4.1.5. Eliminar	15
4.2. Analizar Texto	16
4.3. Configuración	21
5. Log de Errores	23
6. Problemas	26
Referencias	27
A. Freeling	28
B. Reglas Contextuales	30

Capítulo 1

Introducción

El objetivo de este manual es explicar las instrucciones de instalación de la plataforma Lavinia en un servidor web, y además, explicar cómo se utiliza la misma. La aplicación es multiplataforma, es decir, puede ser instalada y configurada en un servidor Linux o Windows. En este manual se encuentran los pasos necesarios para realizar la instalación utilizando los servidores de aplicaciones JBoss [1] y Tomcat [3].

Es necesario tener en cuenta que aunque Lavinia sea multiplataforma, no necesariamente deben serlo sus componentes. Actualmente, Lavinia se distribuye con un conjunto de componentes básicos multiplataforma pero que necesitan algunos datos de configuración. En este manual se especifica la configuración necesaria para cada componente de manera que funcione correctamente.

En este documento se explica cómo pueden integrarse nuevos componentes de análisis a Lavinia. Para esto es necesario ingresar ciertos datos que especifican la función del componente y cumplir con ciertos requerimientos para hacer que el componente implementado en UIMA [4] sea compatible con Lavinia. Además se explica como utilizar cada una de las funcionalidades que provee la plataforma, dando ejemplos y mostrando las páginas web.

En el capítulo 2 se explica la estructura de archivos y directorios de la aplicación. Luego, en el capítulo 3 se explica la instalación de la plataforma y configuración de componentes. En el capítulo 4 se explica cada una de las funcionalidades que provee Lavinia. En el capítulo 5 se explica como configurar el log de errores en el servidor. Por último, en el capítulo 6 se describen algunos errores que podrían ocurrir, y su solución.

Capítulo 2

Estructura de Lavinia

La estructura de archivos y directorios de la aplicación puede verse en la figura 2.1.

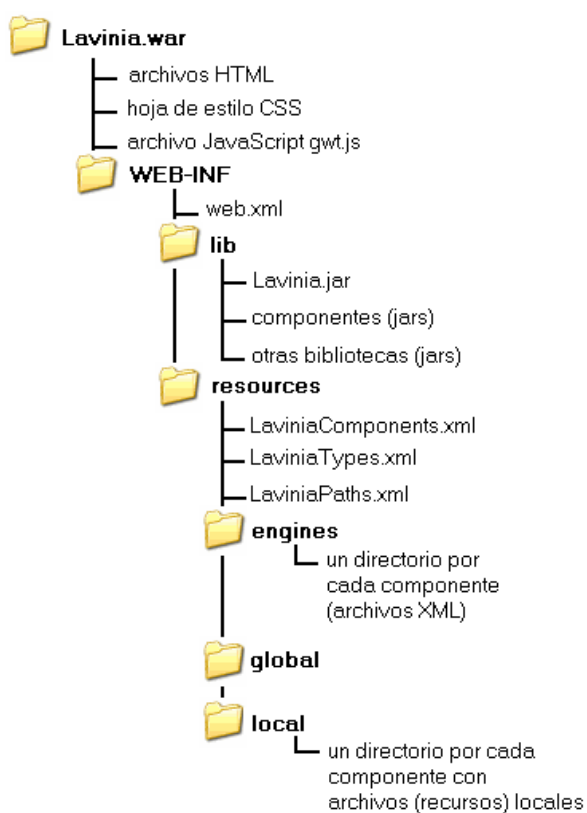


Figura 2.1: Estructura de archivos y directorios

El directorio *lib*, contiene varios archivos JAR [6]. Entre ellos está el archivo *Lavinia.jar* el cual contiene el núcleo de la aplicación. Además, se tiene un JAR por cada componente integrado y otros archivos JAR necesarios para que la aplicación funcione correctamente.

El directorio *resources*, contiene los archivos de configuración y los datos de los componentes de Lavinia:

- El archivo *LaviniaComponents.xml* contiene la lista de los componentes integrados.

- El archivo *LaviniaTypes.xml* contiene la lista de tipos que se pueden tener como entrada o salida de los componentes.
- El archivo *LaviniaPaths.xml* contiene rutas a recursos necesarios para el correcto funcionamiento de sus componentes.

Dentro del directorio *resources*, el directorio *engines* contiene todos los datos de los componentes integrados. Existe un directorio por cada componente, donde se tienen a lo sumo dos archivos XML. Uno de estos archivos siempre va a existir dado que contiene toda la información del componente (parámetros, tipos, recursos, etc.), este archivo va a tener el nombre del componente y consiste en un descriptor como los que utiliza UIMA [4]. Además, puede existir otro archivo XML denominado *configParametersLavinia* el cual contiene información sobre los valores que pueden tomar los parámetros del componente, esto es solo en el caso de tener parámetros que no son multivaluados y tienen un conjunto de valores permitidos.

Además, dentro de *resources* se tienen los directorios *global* y *local*. El primero tiene los recursos (archivos) que son globales, es decir, cualquier componente integrado en la plataforma puede acceder a estos. El segundo tiene un directorio por cada componente donde se guardan los recursos locales a éste, estos archivos sólo pueden ser accedidos por el componente que los contiene.

Capítulo 3

Instrucciones de Instalación

En este capítulo se explica cómo instalar Lavinia en los servidores de aplicaciones JBoss y Tomcat. Al final del capítulo se explica cómo configurar los componentes incluidos en la plataforma luego de haberla instalado en el servidor.

3.1. JBoss

La aplicación fue probada utilizando la versión 4.0.5 del servidor de aplicaciones JBoss. La misma puede descargarse en <http://labs.jboss.com/portal/jbossas/download/index.html>. A continuación se explican los pasos a seguir para instalar la aplicación en dicho servidor.

1. Instalar el servidor JBoss. Sea JBOSS_HOME la ruta del directorio de instalación. Por ejemplo, en Windows, esta ruta por defecto es:
`C:/Archivos de programa/jboss-4.0.5.GA`
2. Para configurar el puerto de escucha de JBoss se debe modificar el archivo `server.xml` del directorio `JBOSS_HOME/server/default/deploy/jbossweb-tomcat55.sar` Donde dice:

```
<!-- A HTTP/1.1 Connector on port 8080 -->  
<Connector port="8080" address="\${jboss.bind.address}"
```

se cambia el puerto 8080 por el que se desee, por ejemplo, lo cambiamos por 80.

3. Descomprimir el archivo `Lavinia.war.zip` en `JBOSS_HOME/server/default/deploy`. Esto dejará el directorio `Lavinia.war` dentro del directorio `deploy` de JBoss.
4. Levantar el servidor utilizando el ejecutable `run` del directorio `JBOSS_HOME/bin`. Es necesario destacar que para poder levantarlo correctamente debe existir una variable de entorno `JAVA_HOME` apuntando a la ruta donde está instalado el *Java Development Kit*.
5. Abrir un cliente web como Internet Explorer [5] o Firefox [7] y escribir la dirección:
<http://localhost:80/Lavinia>

3.2. Tomcat

En lugar de utilizar el servidor JBoss, puede utilizarse Tomcat. El servidor Tomcat es más liviano en el sentido que ocupa menos memoria y se puede levantar y bajar más rápidamente. La versión de Tomcat en la cual se probó Lavinia es la 5.5.20 y puede descargarse en <http://tomcat.apache.org/download-55.cgi>. A continuación se explican los pasos a seguir para instalar la aplicación en dicho servidor.

1. Instalar el servidor Tomcat. Sea TOMCAT_HOME la ruta del directorio de instalación. En Windows, esta ruta por defecto es:
C:/Archivos de programa/Apache Software Foundation/Tomcat 5.5
2. Para configurar el puerto de escucha de Tomcat se debe modificar el archivo `server.xml` del directorio TOMCAT_HOME/conf. Donde dice:

```
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
<Connector port="8080" maxHttpHeaderSize="8192"
  maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
  enableLookups="false" redirectPort="8443" acceptCount="100"
  connectionTimeout="20000" disableUploadTimeout="true" />
```

se cambia el número 8080 por valor de puerto deseado, por ejemplo, lo cambiamos por 80.

3. Descomprimir el archivo `Lavinia.war.zip` en TOMCAT_HOME/webapps. Esto dejará el directorio `Lavinia.war` dentro del directorio `webapps` de Tomcat.
4. Renombrar el directorio `Lavinia.war` obtenido en el paso anterior como `Lavinia` (se elimina la extensión `war`).
5. Levantar el servidor. En Windows esto se hace ejecutando el archivo `tomcat5w.exe` del directorio TOMCAT_HOME/bin.
6. Abrir un cliente web como Internet Explorer o Firefox y escribir la dirección:
<http://localhost:80/Lavinia>

3.3. Configuración

Luego de que la plataforma es instalada en un servidor, al acceder a la misma por primera vez, es decir, al entrar a la página web principal, se realiza una configuración automática. En dicha configuración, se actualizan las rutas a los recursos globales, guardadas en el archivo *LaviniaPaths.xml* y además se actualizan los XML de cada componente (rutas a los recursos globales y locales). Para realizar todos éstos cambios mencionados no se necesita intervención alguna por parte del usuario. Sin embargo, cuando se tienen recursos externos a la plataforma como por ejemplo herramientas (no incluidas dentro de la estructura de directorios de la aplicación), es necesario configurar las rutas a éstas. Para configurar una ruta se selecciona el link «Configuración» en el panel izquierdo de opciones de la página principal, luego se modifica cada ruta que corresponde con una herramienta externa. Cuando se modifica una ruta a un recurso, automáticamente se actualizan todos los componentes que utilizan dicho recurso para que hagan referencia a la nueva ruta. Por más información sobre esta funcionalidad, dirigirse al siguiente capítulo, en la sección «Configuración».

La distribución de Lavinia cuenta con un conjunto de componentes de ejemplo. Por un lado, se tienen componentes que encapsulan parte del conjunto de herramientas de FreeLing. Además, se tiene un componente que trabaja con el formalismo de Reglas Contextuales. La configuración y herramientas externas que necesitan estos componentes para funcionar correctamente se encuentra en los anexos de este documento.

Capítulo 4

Funcionalidades

En este capítulo se explica cómo se utiliza cada una de las funcionalidades que provee la plataforma. El objetivo es explicar cómo se utilizan todas las funcionalidades que se acceden desde web. Se hace especial énfasis en la funcionalidad de «Agregar Componente», dando algunos detalles de cómo implementar un componente utilizando UIMA para luego integrarlo a Lavinia.

En cuanto a la organización del capítulo, se tienen secciones, donde cada sección se corresponde a un link del menú de la página principal de Lavinia.

4.1. Componentes

En el menú de «Componentes» se tiene una lista con los nombres de todos los componentes integrados en el sistema y cinco opciones. En la figura 4.1 se muestra parte de la pantalla.



Figura 4.1: Ver Componentes

4.1.1. Agregar Componente

Como puede verse en la figura 4.2, en la opción «Agregar Componente» se tienen cuatro pestañas: Componente, Parámetros, Tipos y Recursos. A continuación se especifican los datos que se

deben proporcionar al sistema en cada una de las pestañas. Por último, se dan algunas pautas para la implementación del componente utilizando UIMA.

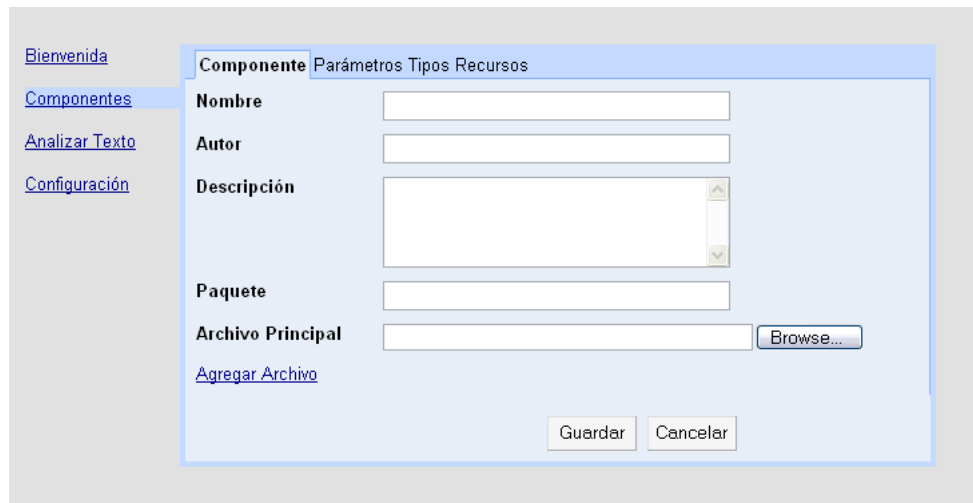


Figura 4.2: Agregar Componente

Pestaña: Componente

Definir un nombre y una descripción para el componente. Además, es necesario especificar el nombre del autor del componente y el paquete en el que se encuentra el archivo Java que lo implementa. En esta pestaña, se agregan también los archivos JAR necesarios, entre estos debe estar el archivo principal el cual debe tener el mismo nombre que el componente (por ejemplo, para el componente *Tokenizador*, su archivo JAR principal es *Tokenizador.jar*). Adicionalmente se pueden agregar otros archivos JAR, en general serán bibliotecas que el componente necesite para su funcionamiento.

Pestaña: Parámetros

Definir que parámetros va a tener el componente, pueden verse como formas de configurarlo. Por ejemplo, un parámetro "Lenguaje" de tipo *String*, un parámetro "DetectarFechas" de tipo *Boolean*, etc.

Para cada parámetro del módulo definir:

- Nombre
- Tipo (*String*, *Float*, *Integer*, *Boolean*, *Archivo*)
- Descripción

Los parámetros de los tipos básicos (todos excepto Archivo), se tratan de la misma forma que los trata UIMA: cada parámetro puede ser requerido o no, y además puede ser multivaluado (tomar un conjunto de valores) o tomar un valor solo. Si es multivaluado se puede dar un conjunto de valores por defecto.

Se tienen dos casos particulares:

- Cuando el componente se implementa en UIMA, se puede definir un parámetro que toma un valor solo (en UIMA, el valor lo debería ingresar el usuario cada vez que quiere utilizar el módulo). En este caso, cuando el componente se integra a Lavinia, se pueden agregar valores por defecto al parámetro (no multivaluado). Estos valores se verán en una lista de opciones cuando el componente se vaya a utilizar para analizar (facilita al usuario, dado que no tiene que escribir el valor). Un ejemplo de esto es el parámetro «Lenguaje» que tienen los componentes que encapsulan FreeLing.
- En Lavinia, se tiene un tipo adicional de parámetro que es el tipo Archivo (o File si se trabaja con la interaz en inglés). Cuando el componente se implementa en UIMA, utiliza archivos que son recursos. Incorporando el componente a Lavinia, se da la opción de que ciertos recursos puedan ser modificados cada vez que el componente se quiere utilizar para analizar, o sea, un archivo que sea un parámetro. Por lo tanto, el archivo en UIMA se trata como un recurso normal, pero en Lavinia, se trata como un parámetro (esto no cambia en nada la implementación del componente). Un ejemplo de esto, son los parámetros que tiene el componente ReglasContextuales.

Pestaña: Tipos

Definir los tipos de anotaciones que el componente requiere como entradas y los tipos nuevos que va a agregar en la salida. El sistema de tipos de Lavinia es el mismo sistema de tipos que define UIMA.

Para cada tipo que el componente utiliza definir:

- Nombre
- Supertipo
- Descripción
- Entrada?
- Salida?

Los tipos se ordenan en una estructura jerárquica por lo que al agregar un tipo es necesario especificar cual es su supertipo. El tipo creado va a heredar todos los atributos del supertipo. El tipo que es superclase de todos los demás y no tiene atributos es *uima.cas.TOP* (es como el tipo *Object* para *Java*). El tipo del cual deben heredar los tipos de anotaciones es *uima.tcas.Annotation*, éste contiene los atributos *begin* y *end*.

Hay dos casos a diferenciar:

- Si se define un tipo cuyo supertipo es distinto de *uima.cas.String*, se pueden agregar **atributos**. Cada atributo consta de un nombre, un tipo, y una descripción. Si el tipo del atributo es *uima.cas.FSArray* o *uima.cas.FSList* se debe especificar cual es el tipo de los elementos que van a estar contenidos en el array o lista respectivamente. El tipo de un atributo y el tipo de elementos de un array o lista, puede ser un tipo predefinido existente en la plataforma o un tipo definido anteriormente para este componente.
- Si se define un tipo cuyo supertipo es *uima.cas.String*, se pueden agregar «valores permitidos» y no se pueden agregar atributos. Por ejemplo, si se va a anotar todas las ocurrencias de la palabra «banco» y se quieren diferenciar los casos en que hace referencia a un mueble, una

entidad financiera, un banco de sangre, etc. Se crea un tipo *A* cuyo supertipo sea *String* tenga como valores permitidos cada una de las opciones mencionadas anteriormente. A su vez, es necesario crear otro tipo *B* cuyo supertipo sea *uima.tcas.Annotation* y tenga un atributo del tipo *A*. El tipo *B* sería una salida del módulo.

Pestaña: Recursos

Definir cuales son los recursos que el componente va a utilizar. Un recurso es un archivo que el componente utiliza. Los archivos pueden ser externos a la aplicación (fuera de su estructura de directorios), en dicho caso se puede tratar de herramientas externas (como *FreeLing*). Los recursos también pueden ser internos a la aplicación (dentro de su estructura de directorios) y en este caso los archivos asociados se deben subir por medio de la página web cuando se agrega el componente. A continuación se detallan las características de cada tipo de recurso y la forma de utilizarlos.

- **Herramientas Externas:** Si el componente va a utilizar una herramienta externa a la plataforma (por ejemplo, *FreeLing*), ésta debe ser configurada previamente, es decir, en el menú «Configuración» de Lavinia se deben agregar las rutas a los archivos que el componente necesita. Luego, cuando se esta integrando el componente, se debe **importar** el recurso asociado a la herramienta.
- **Recursos Internos Globales:** El componente también puede utilizar un recurso interno a la plataforma que es global a todos los componentes. Si el recurso ya existe en la plataforma, cuando se esta integrando el componente, es necesario **importar** dicho recurso. Si el recurso no existe, cuando se integra el componente, se debe **agregarlo** seleccionando el archivo asociado y se debe marcar la opción «Disponible Globalmente» para que el mismo pueda ser importado por otros componentes posteriormente.
- **Recursos Internos Locales:** El componente puede agregar recursos locales, a los cuales sólo él va a poder acceder. Para esto, cuando se está integrando el componente es necesario agregar un recurso seleccionado el archivo asociado. En este caso, la opción «Disponible Globalmente» no debe estar marcada.

Para cada recurso, de cualquier tipo, se puede escribir el nombre de la interfaz que va a manejar dicho recurso y el nombre de la clase que implementa dicha interfaz. Si se especifican estos dos valores, tanto la interfaz como la clase que la implementa deben estar incluidos en alguno de los JARs del componente.

Implementación del Componente

La implementación de los componentes se realiza utilizando la arquitectura UIMA. En este manual de usuario no se pretende explicar cómo se implementan componentes en dicha plataforma. El objetivo es dar una idea y explicar una forma de implementar componentes, por más información se debe consultar el manual de usuario de UIMA disponible en su página web [4].

En primer lugar, se debe implementar una clase que extiende *CasAnnotator_ImplBase*. El nombre de la clase a implementar debe ser el mismo que el nombre del componente que se desea implementar. Se pueden implementar todas las clases adicionales que se consideren necesarias.

La clase tendrá, para cada parámetro del componente, un atributo del tipo correspondiente. Por ejemplo, para el parámetro *Lenguaje*, se tiene un atributo de tipo *String* denominado *Lenguaje*.

La clase tendrá, para cada tipo utilizado un atributo de tipo *Type*. Por ejemplo, un *tokenizador* tiene un atributo de tipo *Type* denominado *Token*.

Procedimientos a implementar:

- `void initialize(UimaContext aContext)`

En este procedimiento se inicializan los valores de los parámetros del componente, los valores se obtienen del contexto que se pasa al procedimiento. Por ejemplo, la siguiente sentencia inicializa el valor del parámetro *lenguaje*:

```
Lenguaje = (String)getContext().getConfigParameterValue('Lenguaje');
```

Además, en este procedimiento se inicializan los recursos, por ejemplo, la siguiente sentencia obtiene el valor del recurso *FreeLingConfEsp*:

```
String freelingConfEsp = getContext()
    .getResourceURL("FreeLingConfEsp").getPath();
```

Nótese que en el caso de parámetros que son archivos (definidos en Lavinia), el valor del parámetro se inicializa como si fuera un recurso y no un parámetro.

- `void typeSystemInit(TypeSystem aTypeSystem)`

En este procedimiento se inicializan los valores de los tipos del componente, según el sistema de tipos definido. Es necesario tener en cuenta el nombre del paquete para los tipos. Por ejemplo, la siguiente sentencia inicializa el valor del atributo *Token* de tipo *Type*:

```
Token = aTypeSystem.getType('com.engines.Token');
```

- `void process(CAS aCAS)`

En este procedimiento se crean/eliminan/modifican anotaciones al CAS pasado como parámetro. Para obtener el texto a analizar se utiliza la siguiente operación:

```
String text = aCAS.getDocumentText();
```

Para crear una anotación de tipo *Token* se utiliza la siguiente operación:

```
AnnotationFS annot = aCAS.createAnnotation(Token, Begin, End);
```

Donde *Begin* y *End* son enteros que indican la región del texto que cubre la anotación (el primer carácter corresponde al índice 0). *Token* es el atributo de la clase (de tipo *Type*). Además, Es necesario agregar la anotación al índice (para que se puedan recorrer las anotaciones):

```
aCAS.getIndexRepository().addFS(annot);
```

Luego de implementado y compilado el componente, se debe crear el archivo JAR principal. Hay varias formas de crear archivos JAR, por ejemplo, si se quiere empaquetar en un JAR el archivo *Tokenizador.java* del paquete *com.engines.Tokenizador*, desde el directorio padre de *com* se escribe en la línea de comandos:

```
jar cvf Tokenizador.jar com/engines/Tokenizador.
```

4.1.2. Agregar Flujo

Mediante esta opción, se pueden guardar en la plataforma flujos predefinidos con valores para sus parámetros. El objetivo de esto es facilitar al usuario, de manera de no tener que seleccionar el

flujo y la configuración de valores para los parámetros cada vez que quiere analizar texto.

Pestaña: Datos

En esta pestaña se debe ingresar nombre y descripción para el flujo, y el nombre del autor del mismo.

Pestaña: Flujo

En esta pestaña se deben seleccionar los componentes que integran el flujo. Es importante destacar que en esta parte, no se realiza ningún chequeo para ver si el flujo es correcto, dado que el flujo podría querer utilizarse sobre resultados importados (ya contienen tipos de anotaciones), por ejemplo.

Pestaña: Parámetros

En esta pestaña se seleccionan los valores para los parámetros del flujo. En el caso de parámetros que son archivos, no se selecciona ningún valor, dado que el archivo deberá subirse al servidor al momento del análisis.

4.1.3. Ver

Al seleccionar un componente y elegir esta opción, se pueden ver todos los datos del mismo. Esta funcionalidad también permite ver la información de los flujos agregados. En el caso de componentes, al igual que en la página donde se agregan, se muestran cuatro pestañas: Componente (nombre, autor, descripción, paquete) , Parámetros (tabla con todos los parámetros), Tipos (tabla con los tipos que maneja) y Recursos (tabla con los recursos que utiliza). En el caso de flujos, se muestran las pestañas: Datos (nombre, autor, descripción), Flujo y Parámetros.

Dado que el objetivo de esta funcionalidad es «ver» los datos, no se permite modificar los valores en los campos ni las tablas. El usuario tiene la opción de volver a la lista de componentes presionando el botón «Atrás» que se encuentra en la parte inferior de la página.

4.1.4. Modificar

Al seleccionar un componente y elegir esta opción, se muestran todos los datos del mismo permitiendo a su vez modificarlos. El formato en el que se muestran los datos es el mismo que se utiliza en la funcionalidad de agregar componente, con la excepción de que no se permite modificar el nombre del componente ni agregar/eliminar archivos JAR. Para el caso de flujos, se permite modificar todos los datos de éstos, excepto el nombre.

Luego de realizar las modificaciones deseadas, el usuario puede guardar los cambios presionando el botón «Guardar» o salir sin guardar los cambios realizados mediante el botón «Cancelar». En ambos casos se retorna a la pantalla anterior donde se tiene la lista de componentes existentes y las opciones.

4.1.5. Eliminar

Esta opción permite eliminar el componente o flujo seleccionado. Al seleccionar un componente o flujo y elegir esta opción, se advierte al usuario y se solicita una confirmación de la acción. En el caso de componentes, se elimina el descriptor del componente, el archivo con información

sobre los parámetros (si existe), los recursos locales, y el archivo JAR principal (con el nombre del componente). Si el componente había agregado o utilizaba recursos globales, estos no se eliminan ya que podrían ser necesarios para otros componentes. Además, si el componente utilizaba otros archivos JAR adicionales, estos no se eliminan ya que la plataforma no guarda registro de los mismos. En el caso de flujos, se elimina únicamente el descriptor XML que guarda la información del flujo y los valores de los parámetros.

ADVERTENCIA: En el caso de componentes, se eliminan los recursos locales que se utilizan, por lo que puede ser necesarios respaldarlos en el servidor antes de eliminarlo de la plataforma.

4.2. Analizar Texto

En el menú «Analizar Texto» se permite al usuario seleccionar un conjunto de componentes, asignarles un orden de ejecución, asignarle valores a los parámetros de los componentes y analizar un texto ingresado en la web o un conjunto de archivos (corpus). Luego de realizar el análisis, se muestran los resultados permitiendo modificar la configuración de colores y los tipos de anotaciones que se van a mostrar, asignándoles un orden de prioridad. Todo el proceso consta de tres pasos bien diferenciados que se explican a continuación.

Primer Paso

En la figura 4.3 se muestra la pantalla correspondiente al primer paso. Por un lado, en el panel izquierdo, se tiene un combo con la lista de todos los componentes y flujos guardados. Al seleccionar un componente o flujo, automáticamente puede visualizarse en la parte inferior, la descripción del mismo, y los tipos de entrada y salida que maneja. Esta información es importante ya que se necesita conocer las entradas y las salidas para poder armar un flujo correcto, es decir, las entradas de cada componente deben estar incluidas en el conjunto de salidas del componente que lo precede en el flujo.

The screenshot displays a web interface titled "Selección de Componentes". On the left, under the heading "Componente", there is a dropdown menu showing "Tokenizador". Below this, the "Descripción" is "tokeniza el texto", "Entradas" is empty, and "Salidas" is "Token". In the center, there are two buttons: ">>" and "<<". On the right, under the heading "Flujo", there is a box containing a list of components: "Tokenizador", "SentenceSplitter", and "POSTagger". To the right of this list are two buttons: "Subir" and "Bajar". At the bottom of the interface, there are two buttons: "Anterior" and "Siguiente".

Figura 4.3: Paso 1: Analizar Texto

El orden de ejecución de los componentes se define en la parte derecha de la pantalla. El usuario debe seleccionar en el combo cada componente, y presionar el botón que lo pasa a la lista de la derecha (flujo). En dicha lista, el componente que está más arriba va a ser el que se ejecuta primero (no debe requerir entradas). Utilizando los botones «Arriba» y «Abajo», el usuario puede cambiar el orden de ejecución de los componentes. Además, se pueden quitar componentes del flujo utilizando el botón con las flechas hacia la izquierda. En el caso en que se seleccione un flujo ya armado, automáticamente se cargan los valores de los parámetros para el mismo.

En el panel de opciones se tiene un link «Cargar Flujo (XML)». Este botón permite que el usuario suba al servidor un archivo que consiste en un descriptor XML con la información del flujo de componentes y valores para los parámetros de los mismos. En el siguiente paso se explica como puede crearse dicho descriptor. Al subir el archivo, automáticamente se carga la lista con los nombres de los componentes ordenados y los valores de los parámetros los cuales se muestran en el paso siguiente.

Al presionar el botón «Siguiente» de la parte inferior de la pantalla, se va al segundo paso del proceso, siempre y cuando el flujo seleccionado no sea vacío y sea correcto.

Segundo Paso

En el segundo paso se realiza la configuración de parámetros de los componentes del flujo. En la figura 4.4 se muestra la pantalla asociada.

Configuración de Parámetros

Componentes

- Tokenizador
- SentenceSplitter
- POSTagger

Componente: POSTagger

Lenguaje: Español

Tagger: HMM

ReTokenizar: Verdadero

PalabrasCompuestas: Verdadero

DetectarNumeros: Verdadero

DetectarPuntuacion: Verdadero

DetectarFechas: Verdadero

DetectarCantidades: Verdadero

Anterior Siguiente

Figura 4.4: Paso 2: Analizar Texto

En la parte izquierda se muestra la lista ordenada de componentes, al presionar en alguno aparece automáticamente un panel a la derecha con los parámetros del mismo. La forma en que se ingresan o seleccionan los valores de los parámetros depende del tipo de parámetro (si es multivaluado, si tiene valores por defecto, etc.). En el caso de parámetros que no son multivaluados y tienen valores por defecto, se muestra un combo donde el usuario debe seleccionar una opción.

Si el parámetro no es multivaluado y no tiene valores por defecto, simplemente se presenta un campo donde el usuario ingresa el texto. En el caso de parámetros multivaluados, se presenta una lista donde se pueden agregar/eliminar valores, si existían valores por defecto, estos ya aparecen agregados a la lista.

En el panel de opciones también aparece un botón «Guardar Flujo (XML)». Este botón es el que permite guardar el flujo de componentes y los valores de los parámetros en un archivo XML, de manera que en otra oportunidad el usuario pueda subir el archivo al servidor (como se explicó en el primer paso).

En este punto, el usuario puede volver al paso anterior para modificar el flujo utilizando el botón «Anterior», o ir al siguiente paso mediante el botón «Siguiente».

Tercer Paso

En el tercer y último paso del proceso se realiza el análisis y se visualizan los resultados. El usuario puede analizar un texto ingresado en la web, para esto el valor del campo «Modalidad» en la parte superior de la página debe ser «Texto». Además, puede analizar un conjunto de archivos (corpus), en este caso el valor del campo «Modalidad» debe ser «Corpus de Texto». Para pasar de un modo a otro, simplemente se elige la opción, y automáticamente la página se actualiza según la modalidad seleccionada.

Modalidad: Texto

En la figura 4.5 se muestra la página en modalidad texto, luego de realizar el análisis.

El usuario debe ingresar el texto a analizar en el campo correspondiente (arriba a la derecha). Luego, el usuario puede analizar el texto ingresado presionando el botón «Analizar». El proceso de análisis puede llevar mucho tiempo, dependiendo del procesamiento que realicen los módulos del flujo. Cuando el proceso se completa, debajo del texto ingresado aparece el resultado del análisis. Si el usuario aún no había modificado la configuración de colores en la tabla, el resultado del análisis va a aparecer sin marcas, con fondo color blanco y letras negras. En la tabla que aparece a la izquierda de la pantalla se define la configuración de colores y tipos de anotaciones que se van a mostrar, mas adelante se explican los detalles.

Modalidad: Corpus

En la modalidad corpus, el usuario debe subir al servidor un conjunto de archivos para analizar. En la figura 4.6 se muestra la página en modalidad corpus de texto.

En la parte izquierda de la página se tiene un panel con forma de «stack», se pueden ver/agregar/eliminar los archivos del corpus, ó se puede cambiar la configuración de colores haciendo click en «Colores». Luego de que el usuario seleccionó los archivos que va a analizar, puede presionar el botón «Analizar» para realizar el proceso. Este proceso puede llevar mucho tiempo, primero se deben cargar los archivos y luego analizarlos, el tiempo del análisis depende del procesamiento que realicen los módulos del flujo. Cuando el proceso termina, pueden verse los resultados de análisis de cada archivo, seleccionándolo y presionando el botón «Ver». Es importante destacar que todos los archivos van a tener la misma configuración de colores y tipos de anotaciones por lo que al cambiar la tabla en un archivo, también se cambiará en los demás del corpus.

Configuración de Colores y Tipos de Anotaciones

Independientemente de la modalidad que se este utilizando (texto o corpus de texto), en ambos casos se tiene una tabla donde se puede modificar la configuración de colores y tipos de anotaciones

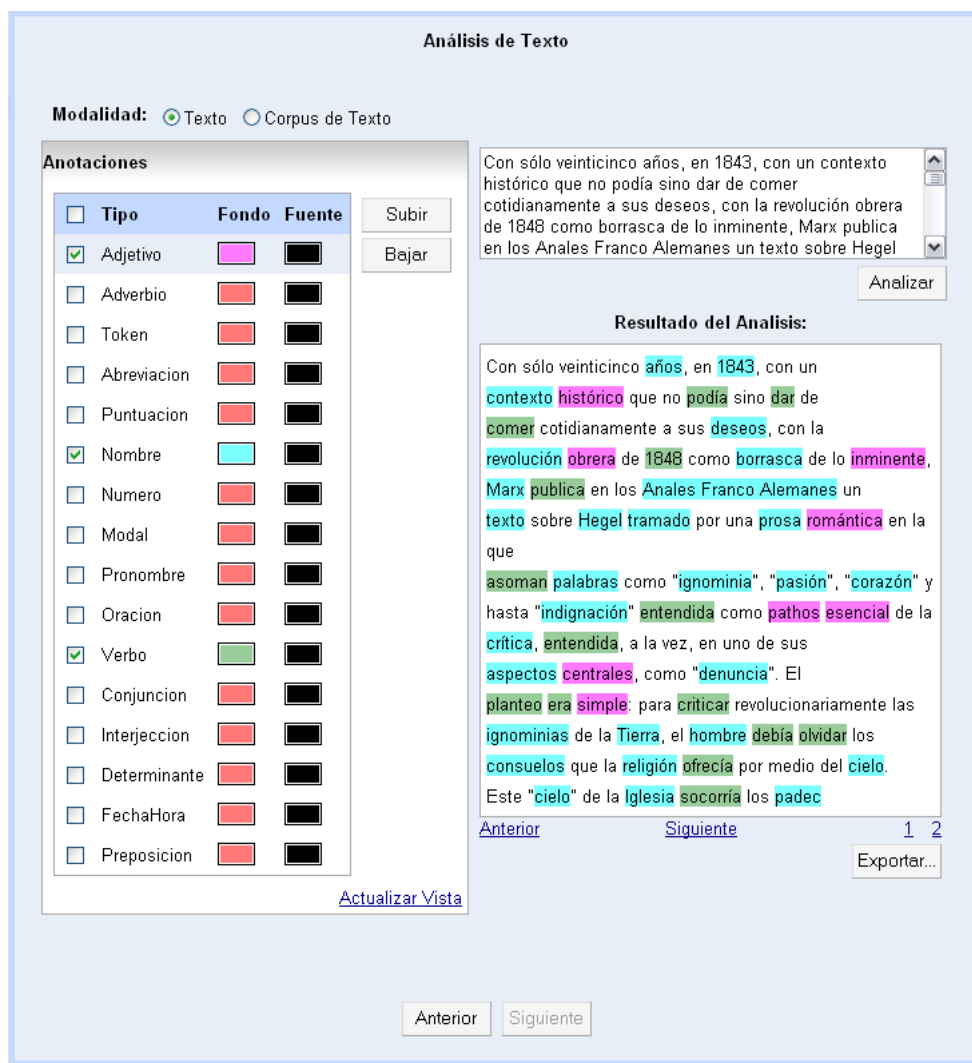


Figura 4.5: Paso 3: Analizar texto ingresado en la web

a mostrar en el resultado de análisis. En la figura 4.5 puede verse dicha tabla en la parte izquierda de la página.

En la tabla se tiene una fila por cada tipo de anotación que tiene como salida el flujo seleccionado. El usuario debe marcar el *checkbox* asociado a cada tipo de anotación que desea visualizar (las que no están marcadas no se visualizan). Por otro lado, el usuario debe asignar un orden de prioridad a los tipos de anotación que desea ver. Esto es importante ya que define cuales anotaciones se van a mostrar primero. Cuanto menos prioridad tenga un tipo de anotación, tendrá más probabilidad de quedar oculta por las demás en caso de solapamiento. Cuánto más arriba esté en la tabla un tipo de anotación, más prioridad tendrá. El orden de las filas se modifica con los botones «Subir» y «Bajar» a la derecha de la tabla. Cada tipo de anotación tiene asociado dos colores, el primero corresponde al color de fondo del texto (columna *Fondo*), y el segundo corresponde al color de la letra (columna *Fuente*).

En las figuras 4.7 y 4.8 pueden verse dos resultados diferentes de visualización para el mismo análisis. A la izquierda de cada figura se muestra la configuración de colores y las prioridades de los

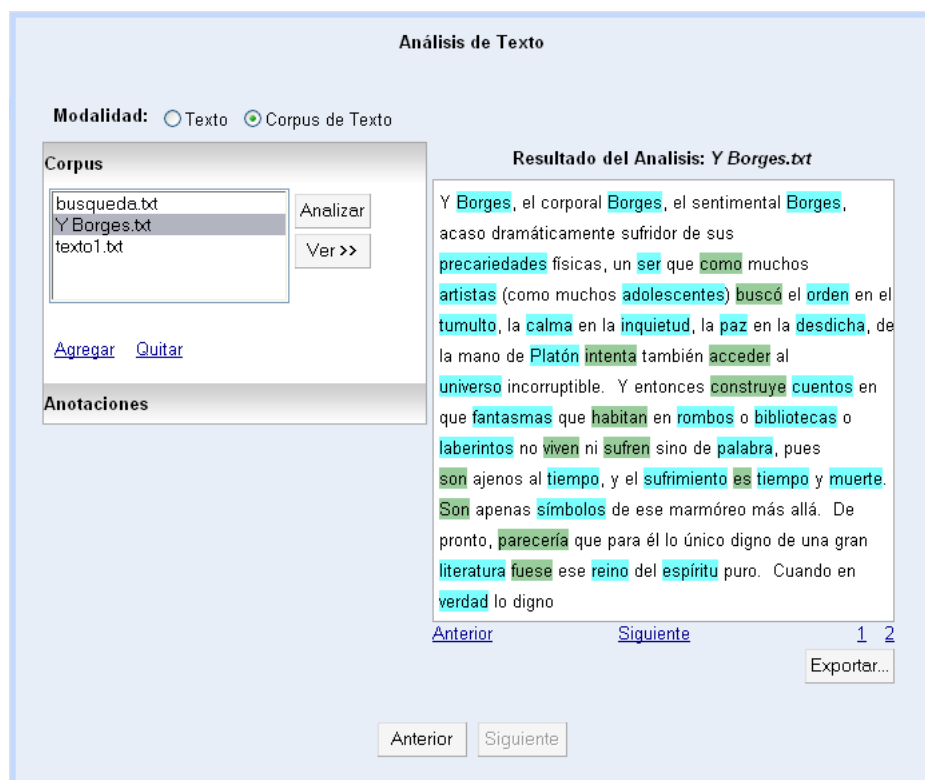


Figura 4.6: Paso 3: Analizar corpus de texto

tipos de anotación. Además, en cada ejemplo se muestra el *popup* que aparece al hacer click sobre una anotación.

Exportación de Resultados

Luego de tener un resultado de análisis, ya sea de texto o de corpus de texto, debajo del resultado se tiene un botón «Exportar...». Al presionar este botón, el usuario puede guardar en su equipo un archivo en formato XMI con los resultados. Es importante destacar que al exportar los resultados, se realiza nuevamente el análisis. Si se está trabajando en la modalidad de corpus de texto, al presionar el botón se exportarán los resultados del archivo que se está visualizando en ese momento.

Importación de Resultados

Si se cuenta con un archivo XMI con resultados de un análisis realizado previamente, se puede subir este archivo al servidor y trabajar con éste mediante la funcionalidad «Importar Resultados» (link en el panel de opciones). Cuando se trabaja con resultados importados, no se realiza ningún chequeo al flujo seleccionado. Por otra parte, es muy importante destacar que si algún tipo de anotación que aparece en los resultados, no está en el flujo seleccionado, este tipo se pierde, es decir, no va a aparecer en los resultados nuevos.

Ejemplo: Se realiza un análisis con el Tokenizador, y se exportan los resultados a un archivo. En otro momento, se importan dichos resultados (subiendo el archivo al servidor), y se selecciona un flujo en el que el único componente es el POSTagger. Este último componente, no funciona por sí solo, ya que necesita las anotaciones de tipo Token como entrada para realizar el análisis. En este caso, dichas anotaciones se encuentran en el archivo XMI guardado, por lo tanto, se podrá realizar el análisis sin problemas.

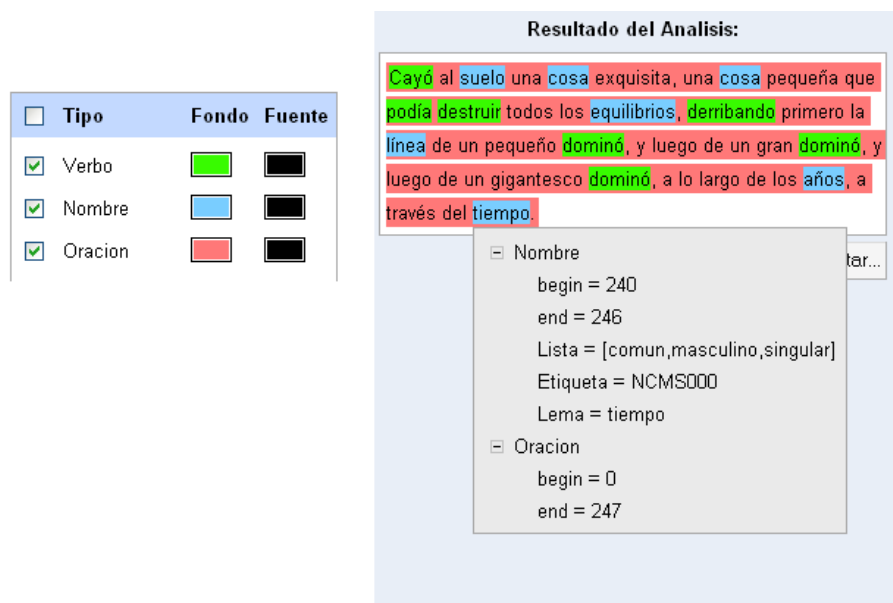


Figura 4.7: Ejemplo 1 de visualización de resultados

4.3. Configuración

En el menú «Configuración», el usuario puede configurar los recursos correspondientes a herramientas externas que se utilizan. En la figura 4.9 se muestra la página de configuración, luego de seleccionar el link «Nuevo». En la página se tiene una tabla donde se muestran los recursos existentes (herramientas externas), para cada recurso se muestra su nombre y la ruta en el sistema de archivos del servidor, además al pasar el puntero del mouse por una fila se muestra la descripción del recurso. A la derecha de la tabla se tienen tres links:

- **Nuevo:** Permite crear un nuevo recurso, ingresando nombre, descripción, y ruta (en el sistema de archivos del servidor).
- **Modificar:** Permite modificar el recurso seleccionado (descripción y ruta).
- **Eliminar:** Permite eliminar el recurso seleccionado (se elimina la información que se tiene del recurso en el sistema, no el archivo asociado a la ruta del recurso).

☐ Tipo
 Fondo
 Fuente

☒ Oracion

☒ Verbo

☒ Nombre

Resultado del Analisis:

Cayó al suelo una cosa exquisita, una cosa pequeña que podía destruir todos los equilibrios, derribando primero la línea de un pequeño dominó, y luego de un gran dominó, y luego de un gigantesco dominó, a lo largo de los años, a través del tiempo.

Oracion

begin = 0
end = 247

Verbo

begin = 195
end = 201
Lista = [principal,indicativo,pasado,tercera persona,singular]
Etiqueta = VMIS3S0
Lema = dominar

Figura 4.8: Ejemplo 2 de visualización de resultados

[Bienvenida](#)
[Componentes](#)
[Analizar Texto](#)
[Configuración](#)

Configuración de Recursos

Recurso	Ruta
FreeLingExe	C:/FreeLing-1.4/analyzer.exe
FreeLingConfEsp	C:/FreeLing-1.4/data/config/es.cfg
FreeLingConfEng	C:/FreeLing-1.4/data/config/en.cfg
FreeLingConfIt	C:/FreeLing-1.4/data/config/it.cfg
FreeLingConfCat	C:/FreeLing-1.4/data/config/ca.cfg
EditorRC	Z:/editorRC/paraLavinia.pl
PrologExe	C:/Program Files/pl/bin/plcon.exe

Nuevo

Modificar

Eliminar

Nombre
 Descripción
 Ruta

Browse...

Aceptar

Cancelar

Figura 4.9: Configuración de Recursos

Capítulo 5

Log de Errores

Lavinia utiliza Log4j [9] para el log de errores. La primera y una de las mayores ventajas de cualquier API de logging sobre el tradicional `System.out.println` es la capacidad de habilitar y deshabilitar ciertos logs, mientras otros no sufren ninguna alteración. Esto se realiza categorizando los mensajes de logs de acuerdo a cierto criterio. Log4J tiene por defecto 5 niveles de prioridad para los mensajes de logs:

- **DEBUG**: Se utiliza para escribir mensajes de depuración, este log no debe estar activado cuando la aplicación se encuentre en producción.
- **INFO**: Se utiliza para mensajes similares al modo «verbose» en otras aplicaciones.
- **WARN**: Se utiliza para mensajes de alerta sobre eventos que se desea mantener constancia, pero que no afectan el correcto funcionamiento del programa.
- **ERROR**: Se utiliza en mensajes de error de la aplicación que se desea guardar, estos eventos afectan al programa pero lo dejan seguir funcionando, como por ejemplo que algún parámetro de configuración no es correcto y se carga el parámetro por defecto.
- **FATAL**: Se utiliza para mensajes críticos del sistema, generalmente luego de guardar el mensaje el programa abortará.

Adicionalmente a estos niveles de log, existen 2 niveles extras que solo se utilizan en el archivo de configuración, estos son:

- **ALL**: este es el nivel más bajo posible, habilita todos los logs.
- **OFF**: este es el nivel más alto posible, deshabilita todos los logs.

Log4j permite que los mensajes de logs se impriman en multiples destinos; en Log4J el destino de salida se denomina un *appender*. Algunos de los Appenders disponibles son:

- *ConsoleAppender* `org.apache.log4j.ConsoleAppender`: Despliega el log en la consola.
- *FileAppender* `org.apache.log4j.FileAppender`: Redirecciona los mensajes de logs hacia un archivo.
- *RollingFileAppender* `org.apache.log4j.RollingFileAppender`: Redirecciona los mensajes de logs hacia un archivo, se pueden definir políticas de rotación para que el archivo no crezca indefinidamente.

- *DailyRollingFileAppender* `org.apache.log4j.DailyRollingFileAppender`: Redirecciona los mensajes de logs hacia un archivo, se pueden definir políticas de rotación basados en la fecha.
- *SocketAppender* `org.apache.log4j.net.SocketAppender`: Redirecciona los mensajes de logs hacia un servidor remoto de log.
- *SMTPAppender* `org.apache.log4j.net.SMTPAppender`: Envía un mail con los mensajes de logs, típicamente se utiliza para los niveles ERROR y FATAL.

Otra de las características importantes de Log4j es el *layout*. El layout es responsable de formatear los mensajes de logs de acuerdo a las definiciones del desarrollador. Los tipos de Layouts disponibles son:

- *SimpleLayout*: Consiste de la prioridad del mensaje, seguido por `%z` luego el mensaje de log.
- *PatternLayout*: Especifica el formato de salida de los mensajes de logs de acuerdo a unos patrones de conversión similares a los utilizados en la función `printf` en lenguaje C.
- *HTMLLayout*: Especifica que la salida de los eventos será en una tabla HTML.
- *XMLLayout*: Especifica que la salida será a un archivo XML que cumple con el `log4j.dtd`.
- *TTCCLayout*: Consiste de la fecha, thread, categoría y NDC, cualquiera de estos cuatro campos pueden ser habilitados y deshabilitados individualmente.

En el archivo de configuración del log se define el nivel log que se quiere utilizar, el appender y el layout (entre otras cosas). Este archivo se encuentra en el directorio *resources* de Lavinia, se denomina `log4j.properties`. En los cuadros 1 y 2, pueden verse dos ejemplos de archivos de configuración. En el primer caso, se trata de un archivo de configuración para enviar los mensajes del log a la consola. En el segundo caso, los mensajes se envían a un archivo.

Cuadro 1 Archivo de Configuración para redireccionar mensajes a consola

Coloca el nivel del root logger en ALL y adiciona un solo appender que es A1.
log4j.rootLogger=ALL, A1

A1 es configurado para utilizar ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender

A1 utiliza PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=[%5p] %d{mm:ss} (%F:%M:%L)%n%m%n%n

Cuadro 2 Archivo de Configuración para redireccionar mensajes a archivo

Coloca el nivel del root logger en ALL y adiciona un solo appender que es A2.
log4j.rootLogger=ALL, A2

A2 es configurado para utilizar FileAppender.
log4j.appender.A2=org.apache.log4j.FileAppender
log4j.appender.A2.File=c:/app_a2.log

A2 utiliza PatternLayout.
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
log4j.appender.A2.layout.ConversionPattern= [%5p] %d{mm:ss} (%F:%M:%L)%n%m%n%n

Capítulo 6

Problemas

En este capítulo se presentan algunos de los problemas que pueden ocurrir al utilizar la plataforma, y la forma de resolverlos.

1. **Problema:** Al entrar a la página principal aparece el mensaje «Error al obtener rutas a archivos».

Solución: Este problema ocurre cuando no se puede obtener las rutas a los archivos de la plataforma utilizando la información que proporciona el servidor. En general, el problema se debe a que los nombres de los directorios fueron modificados, en particular, el directorio «WEB-INF» debe tener su nombre en letras mayúsculas, ya que si está en minúsculas no se puede encontrar la información y el sistema despliega dicho error.

2. **Problema:** Al intentar realizar un análisis, aparece el mensaje «Error al inicializar recursos del flujo».

Solución: Este error se debe a que las rutas a los recursos que utiliza algunos de los módulos del flujo están mal configuradas. Puede ser que solucione modificando las rutas a los recursos en el menú «Configuración», para que sean las rutas correctas. De lo contrario, puede ser necesario modificar la información que guardan los componentes sobre los recursos que utilizan, ya que podrían estar haciendo referencia a un recurso que fue eliminado del sistema. En el caso en que el mensaje de error haga referencia a un recurso «DYNAMIC», se trata de un parámetro de un componente que consiste en un archivo y no fue seleccionado antes de intentar realizar el análisis.

3. **Problema:** Al intentar realizar un análisis, aparece el mensaje «Error al realizar procesamiento».

Solución: Este error ocurre cuando ocurrió una excepción en el código de alguno de los módulos del flujo. La solución depende de los módulos de análisis, en general si estos utilizan un log de errores se sugiere consultarlo. Si los módulos utilizan herramientas externas, puede que sea un problema de configuración de las mismas, para solucionar esto se modifican las rutas a los recursos en el menú «Configuración».

Referencias

- [1] JBoss A division of RedHat. Jboss Application Server. <http://labs.jboss.com/portal/jbossas>. (Último acceso: enero de 2007).
- [2] FreeLing. An Open Source Suite of Language Analyzers. <http://garraf.epsevg.upc.es/freeling/>. (Último acceso: enero de 2007).
- [3] The Apache Software Foundation. Apache Tomcat. <http://tomcat.apache.org/>. (Último acceso: enero de 2007).
- [4] IBM. Uima, unstructured information management architecture. <http://www.research.ibm.com/UIMA/>. (Último acceso: febrero de 2007).
- [5] Microsoft. Internet Explorer. <http://www.microsoft.com/windows/ie/>. (Último acceso: enero de 2007).
- [6] Sun Microsystems. JAR File Specification. <http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html>. (Último acceso: mayo de 2006).
- [7] Mozilla. Firefox. <http://www.mozilla.com/en-US/>. (Último acceso: enero de 2007).
- [8] Prolog. What is swi-prolog? <http://www.swi-prolog.org/>. (Último acceso: noviembre de 2007).
- [9] Logging Services. Log4j. <http://logging.apache.org/log4j/docs/>. (Último acceso: mayo de 2007).

Anexo A

Freeling

Existen cuatro componentes que utilizan el conjunto de herramientas de FreeLing:

- **Tokenizador:** Detecta los tokens en el texto, creando una anotación diferente para cada uno. Tiene un único parámetro de configuración que permite al usuario especificar el idioma del texto a analizar (Inglés, Español, Catalán e Italiano). No se requiere ningún tipo de anotación como entrada para realizar el análisis.
- **SentenceSplitter:** Detecta las oraciones del texto. Al igual que el módulo que tokeniza el texto, se tiene un único parámetro para configurar el idioma del texto y no requiere tipos de anotación como entrada. El módulo crea una anotación diferente para cada oración encontrada.
- **POSTagger:** Realiza el etiquetado del texto, según la categoría gramatical, éste módulo es el POSTagger. Este componente requiere como entrada las anotaciones que realiza el Tokenizador, y genera como salida anotaciones de distintos tipos según la categoría (Nombre, Verbo, Adjetivo, Puntuación, etc.). Tiene un conjunto de parámetros de configuración, definidos por FreeLing. La mayoría de estos parámetros son para configurar si el módulo debe detectar cierto elemento, por ejemplo, números, fechas, etc. Además tiene un parámetro para configurar el tipo de etiquetador que se va a usar, por ejemplo el que utiliza modelos de Markov ocultos.
- **FreeLingAnnotator:** Este módulo realiza la misma tarea que el POSTagger y tiene los mismos requerimientos de entrada y parámetros de configuración. La diferencia es que el FreeLingAnnotator no crea una anotación diferente para cada categoría gramatical, tiene un sólo tipo de salida, el cual tiene atributos con la información de la categoría, lema, etc.

Para que los componentes funcionen correctamente es necesario instalar FreeLing, esta herramienta puede descargarse gratuitamente desde su página oficial [2]. Los componentes fueron probados utilizando las versiones 1.4 y 1.5 de *Freeling* en Windows y Linux.

Para poder utilizar *Freeling*, es necesario contar con un archivo de configuración por cada idioma. Los idiomas disponibles son Inglés, Español, Italiano, y Catalán. Los archivos de configuración para cada idioma deben tener un nombre específico según se detalla a continuación:

Español = es.cfg

Catalán = ca.cfg

Inglés = en.cfg

Italiano = it.cfg

Al instalar *Freeling*, estos archivos se crean automáticamente pero es imprescindible modificar las rutas que se encuentran en éstos para que sean las correctas.

Luego de tener *Freeling* instalado, se debe realizar la configuración de dicha herramienta en Lavinia. Para esto, es necesario modificar los siguientes recursos (menú «Configuración»):

- *FreeLingConfEsp*: Este recurso especifica la ruta al directorio que contiene el archivo de configuración para el idioma español, por ejemplo, la ruta podría ser:
`c:/FreeLing-1.4/data/config/es.cfg`.
- *FreeLingConfCat*: Este recurso especifica la ruta al directorio que contiene el archivo de configuración para el idioma catalán.
- *FreeLingConfEng*: Este recurso especifica la ruta al directorio que contiene el archivo de configuración para el idioma inglés.
- *FreeLingConfIt*: Este recurso especifica la ruta al directorio que contiene el archivo de configuración para el idioma italiano.
- *FreeLingExe*: Este recurso especifica la ruta al archivo ejecutable de *Freeling*, en general este archivo se denomina **analyzer**. Por ejemplo, la ruta podría ser:
`c:/FreeLing-1.4/analyzer.exe`.

Es muy importante que en las rutas a los archivos se incluya el nombre completo de los mismos, es decir, incluyendo su extensión.

Anexo B

Reglas Contextuales

El componente de reglas contextuales que se distribuye con Lavinia necesita tener instalado Prolog [8] y además un conjunto de archivos de Prolog que consisten en la lógica del componente.

Luego de haber instalado Prolog y de tener los archivos prolog de la lógica del componente en un mismo directorio, se puede pasar a configurar estos recursos en Lavinia (menú «Configuración»). Los recursos o herramientas externas a configurar son los siguientes:

- *PrologExe*: Ruta al archivo ejecutable de Prolog. Por ejemplo, en Windows, esta ruta podría ser: `C:/Archivos de Programa/pl/bin/plcon.exe`. En Linux, la ruta podría ser, por ejemplo: `/usr/local/bin/pl`
- *EditorRC*: Ruta al archivo "paraLavinia.pl" que utiliza el componente. Este archivo debe estar en el mismo directorio donde se encuentran los demás archivos de prolog. Por ejemplo, la ruta en Windows podría ser `Z:/editorRC/paraLavinia.pl`

El componente requiere como entrada el tipo de anotación que tienen como salida el FreeLingAnnotator, por lo tanto, el flujo correcto debería ser:

Tokenizador – >FreeLingAnnotator – >ReglasContextuales.

Como salida, se genera un sólo tipo de anotación que contiene la etiqueta que se asignó (según la regla aplicada) y la lista de atributos.

Se tienen dos parámetros de configuración que consisten en archivos. Estos archivos se deben subir al servidor cada vez que el componente se quiera utilizar para analizar. El primer parámetro, denominado «Reglas», consiste en el archivo de texto plano que contiene las reglas contextuales a aplicar en el análisis. El segundo parámetro, denominado «Condiciones» consiste en un archivo de prolog con hechos.