Lavinia: Ambiente Web para PLN

Informe de Proyecto de Grado

Cecilia Techera

Tutores:
MSc Diego Garat,
MSc Guillermo Moncecchi

Instituto de Computación,
Facultad de Ingeniería,
Universidad de la República,
Montevideo, Uruguay.

Agosto, 2007

Resumen

Actualmente, existe una gran cantidad de herramientas de procesamiento de textos (*tokenizadores*, analizadores morfológicos, analizadores sintácticos, etc.). Estas herramientas, en general, no siguen un estándar común para la representación del texto y los resultados.

Por otra parte, en el área de procesamiento de lenguaje natural existen muchas personas involucradas que no son necesariamente del área informática, por ejemplo lingüistas. Estas personas no tienen necesidad de saber detalles de implementación de las herramientas ni de la representación estándar del análisis, sin embargo, necesitan poder utilizarlas.

En este proyecto se pretende, en primer lugar, desarrollar un ambiente para el procesamiento de lenguaje natural. Adicionalmente, interesa desarrollar la interfaz de usuario del ambiente utilizando el enfoque *Web 2.0*. El proyecto pretende contar con una interfaz amigable donde el usuario se sienta «cómodo» y pueda interactuar de manera sencilla con la aplicación.

En este ambiente se aspira a la integración de módulos de análisis que encapsulen herramientas existentes, permitiendo el encadenamiento de módulos para realizar análisis complejos. Además, se deberá definir un estándar común para la representación de textos.

Se desarrolla un sistema denominado *Lavinia*, basado en *UIMA*, una de las plataformas ya existentes. *Lavinia* sigue una arquitectura cliente-servidor donde la interfaz de usuario es web y desarrollada utilizando el marco de trabajo *Google Web Toolkit*.

Como aporte importante, se destaca la visualización de resultados implementada. Para su desarrollo se tuvieron en cuenta ciertos problemas como el solapamiento de anotaciones.

Se integraron módulos de análisis a *Lavinia* que cumplen algunas de las principales tareas de procesamiento de lenguaje natural. Los módulos integrados encapsulan algunos de los análisis que realiza *FreeLing*.

Palabras Clave: Ambiente, FreeLing, Google Web Toolkit, Procesamiento de Lenguaje Natural, UIMA, Web 2.0.

Dedicatoria y Agradecimientos

Dedico este proyecto principalmente a mi familia, por haberme apoyado y ayudado en todo los posible durante los años de mi carrera. A mi novio, por siempre estar ahí tantas veces que lo necesité, por ser mi mejor amigo. A mis amigas, por su apoyo incondicional.

Agradezco a mis tutores, Diego y Guillermo, quienes siempre me guiaron y tuvieron una disposición excelente desde todo punto de vista. Gracias por el buen humor que siempre han tenido y por estar siempre ahí.

Tabla de Contenidos

Capítulo 1

Introducción

El área de Procesamiento de Lenguaje Natural (PLN) se encarga de facilitar la comunicación hombre-computadora y para ello, necesita estudiar el lenguaje y sus fenómenos de manera de poder entender cómo realizarla. Existe una amplia variedad de aplicaciones de PLN. Por ejemplo, la recuperación de información se encarga de retornar un conjunto de documentos a partir de una consulta específica, como el buscador *Google*. Otra aplicación del PLN más relacionada a la interacción entre el hombre y la computadora es el reconocimiento de voz.

En la actualidad, existe una enorme cantidad de información de todo tipo, disponible para todo el mundo a través de Internet. Esta información se puede encontrar en diferentes formatos como ser audio, texto, video, etc. Las aplicaciones de PLN utilizan toda esta información como una gran fuente de conocimiento, pero antes de ser utilizada se debe procesar. Uno de los retos de PLN es estructurar y analizar la información para poder interpretar, detectar y localizar conceptos de interés que no están explícitamente etiquetados o señalados y sus relaciones. Los resultados del análisis deben ser puestos de una forma estructurada de manera que tecnologías de búsqueda, minería de datos, motores de bases de datos, etc., puedan encontrar eficientemente los conceptos que se necesitan [14].

El procesamiento que se realiza sobre la información generalmente se puede dividir en varias etapas. Cada etapa consiste en cierto análisis que se basa en los análisis realizados en etapas previas. Desde el punto de vista del *software*, podemos ver cada etapa como un módulo diferente. Los módulos forman una cadena o flujo, y la información va pasando por cada uno de ellos, hasta que llega al final de la cadena donde se obtiene el resultado.

En particular, en este proyecto nos focalizaremos en procesamiento de texto. Generalmente, existe una forma de representar el texto

y los resultados de su análisis, una estructura que todos los módulos de la cadena van a utilizar para agregar sus resultados de análisis o incluso modificar los resultados a los que llegaron los módulos anteriores.

Actualmente, hay una gran cantidad de herramientas de procesamiento de textos (*tokenizadores*, analizadores morfológicos, analizadores sintácticos, etc.). Estas herramientas, en general, no siguen un estándar común para la representación del texto y los resultados. Las personas que desarrollan estas herramientas definen una representación y la utilizan. El hecho de que no exista un estándar común, dificulta la tarea de encadenar herramientas para realizar un análisis completo. A modo de ejemplo, si se quiere *tokenizar* un texto usando cierta herramienta H_1 y luego utilizar el resultado para que la herramienta H_2 etiquete el texto realizando análisis morfológico, es necesario «traducir» la salida de H_1 al formato de representación que admite como entrada H_2.

Por otra parte, en el área de procesamiento de lenguaje natural existen muchas personas involucradas que no son necesariamente del área informática, por ejemplo lingüistas. Estas personas no tienen necesidad de saber detalles de la implementación de un módulo ni de la representación estándar del análisis, sin embargo, necesitan poder utilizar estas herramientas. Debe ser fácil para un usuario seleccionar un conjunto de herramientas a utilizar, asignarles un orden en el que se ejecutarán y utilizarlas en cadena para analizar un texto o un conjunto de textos. Además, el usuario debe entender los resultados de análisis de forma fácil y rápida, sin necesidad de conocer la forma en que se representan.

En la actualidad, es cada vez más común el desarrollo de aplicaciones web. Una de las ventajas de este tipo de aplicaciones es que el usuario no necesita instalar el *software*, el mismo está disponible simplemente accediendo a una página desde cualquier computador. Además, es muy fácil hacer que la aplicación quede accesible desde cualquier parte del mundo. Actualmente, la web ha pasado a una nueva generación denominada web 2.0. Las aplicaciones web tradicionales no hacían tanto énfasis en la satisfacción del usuario como lo hacen las de la nueva generación.

En este proyecto se pretende, en primer lugar, desarrollar un ambiente para el procesamiento de lenguaje natural. Este ambiente se enfoca principalmente a la utilidad de manera de permitir a usuarios tanto del área informática como de otras áreas, el acceso a un conjunto de funcionalidades para el análisis de texto. Si bien en la actualidad ya existen aplicaciones con estas características, el proyecto pretende el desarrollo de una aplicación más orientada al

usuario y a su satisfacción.

Adicionalmente, interesa desarrollar la interfaz de usuario del ambiente utilizando el enfoque web 2.0. El proyecto pretende contar con una interfaz amigable donde el usuario se sienta «cómodo» y pueda interactuar de manera sencilla con la aplicación. En nuestro caso en particular, el desarrollo de una aplicación web es de suma importancia, ya que permitiría que cualquier persona pueda integrar sus propios módulos de análisis a la plataforma y utilizarlos encadenados con otros ya existentes para analizar texto.

El ambiente mencionado será la interfaz de usuario de la plataforma a desarrollar. En esta plataforma se aspira a la integración de módulos de análisis que encapsulen herramientas existentes, permitiendo el encadenamiento de módulos para realizar análisis complejos. Además, se deberá definir un estándar común para la representación de textos. Este estándar, será utilizado en el futuro cuando se desarrollen nuevas herramientas, de manera de eliminar la dificultad existente al tratar de utilizar herramientas que no utilizan la misma representación.

1.1. Objetivos

El primer objetivo consiste en estudiar el área del procesamiento de lenguaje natural, investigando estándares para representación de texto, herramientas de procesamiento de lenguaje natural y arquitecturas para su organización. Se deberán estudiar los sistemas de procesamiento de lenguaje natural existentes, teniendo en cuenta la representación de texto que utilizan, arquitectura general, cómo se pueden integrar nuevos módulos de análisis, visualización de resultados, etc.

Como segundo objetivo, se debe definir un estándar para la representación de textos de manera de poder representar el texto y los resultados de su análisis. Este estándar de representación será utilizado por los módulos de análisis de la plataforma y será el que se utilice en el futuro cuando se desarrollen nuevas herramientas.

Como tercer objetivo, se tiene el desarrollo de una plataforma amigable y de fácil uso para la selección y ordenamiento de herramientas a aplicar sobre un texto en lenguaje natural. El usuario debe poder seleccionar un conjunto de herramientas, asignarles un orden, y utilizarlas para analizar un texto o un conjunto de textos.

El cuarto objetivo del proyecto consiste en la implementación de la visualización de resultados de análisis, de forma de poder interpretarlos de manera sencilla.

Por último, se tiene como objetivo desarrollar módulos para integrar a la nueva plataforma que cumplan con las principales tareas de procesamiento de lenguaje natural (*tokenización*, análisis mofológico, análisis sintáctico, etc.).

1.2. Organización del Documento

Habiendo planteado y motivado el problema, el resto del documento se organiza de la siguiente forma:

En el capítulo 2, se presenta un resumen del estado del arte. Se da una breve introducción al procesamiento y análisis de texto, y se describen algunas de las herramientas y arquitecturas existentes relacionadas al proyecto.

En el capítulo 3 se describe Lavinia, la aplicación desarrollada, en términos de las funcionalidades que provee.

En el capítulo 4, se describe el diseño e implementación de la aplicación desarrollada. En primer lugar se explica la arquitectura del sistema y sus componentes. Luego, se presentan las herramientas y técnicas utilizadas para la implementación. Se describe el diseño y la implementación del algoritmo para la visualización de resultados de análisis.

Por último, en el capítulo 5, se establecen las conclusiones del trabajo, se destacan los aportes realizados y se mencionan las mejoras a implementar.

Capítulo 2

Estado del Arte

En este capítulo se presenta una revisión del estado del arte en el tema del trabajo. En primer lugar, se presentan algunos conceptos generales, necesarios para entrar en contexto con el proyecto. Luego, se describen los sistemas de PLN en general, presentando algunas características particulares de los sistemas que se estudiaron. Principalmente, se estudiaron sus arquitecturas de organización, los formatos de representación del texto que utilizan y la visualización de resultados de análisis que ofrecen. El principal objetivo de este estudio fue evaluar qué tan razonable era el diseño e implementación de una nueva plataforma de PLN teniendo en cuenta que ya existen varias en la actualidad. Es muy importante evaluar las ventajas y desventajas de cada una y determinar si alguna cumple con los objetivos del proyecto.

2.1. Conceptos Generales

Una de las ramas más importantes de la Inteligencia Artificial es aquella orientada a facilitar la comunicación hombre-computadora por medio del lenguaje humano, o lenguaje natural. El Procesamiento del Lenguaje Natural (PLN) es la disciplina encargada de producir sistemas informáticos que posibiliten dicha comunicación. Se trata de una disciplina tan antigua como el uso de las computadoras (años 50), de gran profundidad, y con aplicaciones tan importantes como la traducción automática o la búsqueda de información en Internet [16].

Otra de las aplicaciones consiste en realizar resúmenes. Esta tarea presenta muchas dificultades. Por ejemplo, si se pueden determinar cuáles son las oraciones más importantes de un texto, puede pasar que en esas oraciones se haga referencia a objetos, eventos, personas,

etc., definidos en las oraciones que no se consideraron importantes. Por lo tanto, no alcanza sólo con extraer las oraciones importantes, se requiere un análisis mucho más detallado.

Una de las aplicaciones del PLN que más éxito ha obtenido en la actualidad es la de reconocimiento de voz, ya que las computadoras de hoy tienen esta característica. El reconocimiento de voz puede tener dos posibles usos: para identificar al usuario o para procesar lo que el usuario dicte, existiendo ya programas comerciales que son accesibles por una gran cantidad de personas. Además de las aplicaciones mencionadas, existen muchas más [12].

Las aplicaciones de PLN necesitan estudiar el lenguaje natural en profundidad y para esto utilizan distintos tipos de análisis. El estudio del lenguaje natural se estructura normalmente en cuatro niveles: morfológico, sintáctico, semántico y pragmático [7].

En el nivel morfológico, se detecta la relación que se establece entre las unidades mínimas que forman una palabra (morfemas), como puede ser el reconocimiento de sufijos o prefijos.

El análisis sintáctico tiene como función etiquetar cada uno de los componentes sintácticos que aparecen en la oración y analizar cómo las palabras se combinan para formar construcciones gramaticalmente correctas. El resultado de este proceso consiste en generar la estructura correspondiente a las categorías sintácticas formadas por cada una de las unidades léxicas que aparecen en la oración.

El análisis semántico trata del significado. Definir qué es el significado no es una tarea sencilla, y puede dar lugar a diversas interpretaciones. Es posible distinguir entre significado independiente y significado dependiente del contexto. El primero, tratado por la semántica, hace referencia al significado que las palabras tienen por sí mismas sin considerar el significado adquirido según el uso en una determinada circunstancia. La semántica, por tanto, hace referencia a las condiciones de verdad de la frase, ignorando la influencia del contexto o las intenciones del hablante.

Por otra parte, el análisis pragmático estudia el componente significativo de una frase asociado a las circunstancias en que ésta se da. Este significado es el que se conoce como dependiente del contexto.

Además se pueden incluir otros niveles de conocimiento como es la información fonológica, referente a la relación de las palabras con el sonido asociado a su pronunciación; el análisis del discurso, que estudia cómo la información precedente puede ser relevante para la comprensión de otra información; y, finalmente, lo que se denomina conocimiento del mundo, referente al conocimiento general que los hablantes han de tener sobre la estructura del mundo para mantener una conversación [7].

Muchos de estos análisis se utilizan encadenados. Por ejemplo, para realizar el análisis semántico se necesitan los resultados del análisis sintáctico y a su vez el análisis sintáctico utiliza los resultados del análisis mofológico. Cada uno de estos análisis necesita el resultado de los análisis anteriores. Además, es posible que se necesiten recursos como diccionarios, gramáticas, etc.

Desde el punto de vista del *software*, podemos asociar cada análisis a un módulo diferente. Entonces, cada módulo de análisis debe utilizar una estructura para representar sus resultados. Lo deseable es que esta estructura siga algún estándar común y sea compartida por todos los módulos de la cadena. Cada módulo agregará sus resultados e incluso podrá modificar o eliminar elementos generados por los módulos que lo preceden.

Cuando se analiza texto, el resultado del análisis generalmente se representa mediante *anotaciones* o etiquetas. Una operación de etiquetado o anotación se basa en un conjunto de etiquetas que representan las diversas categorías lingüísticas o extra-lingüísticas. Una clasificación posible para las anotaciones es la siguiente:

- Anotaciones con información textual o extra-textual (por ejemplo, lenguaje, origen, autor, etc).
- Anotaciones con información ortográfica.
- Anotaciones con información lingüística.

Entre las anotaciones con información lingüística se destacan unidades léxicas (nombres propios, abreviaturas, etc), anotaciones morfosintácticas (asignan a cada unidad léxica una etiqueta o referencia a una entidad), lematización (reducción de las palabras flexionadas a su forma base, su lema), constituyentes sintácticos (representación de los árboles sintácticos, etc), caracterización semántica (desambigüación de unidades homógrafas, etc), anotaciones de discurso (estructura de textos, párrafos, títulos, etc) [21].

A modo de ejemplo, podría tenerse un módulo de análisis que se encargue de detectar nombres de personas, lugares, fechas, etc. Dada la oración «Woody Allen llegó a Donosti el miércoles a las dos», un análisis utilizando dicho módulo identificaría las anotaciones que se muestran en la figura 2.1. Cada anotación se muestra de un color diferente, y debajo se muestran sus características o atributos.

Las anotaciones pueden tener atributos, por ejemplo, un análisis que busca todos los nombres propios de personas en un texto (María, Pedro, etc.), podría etiquetar o anotar cada nombre y además, a cada anotación agregarle un atributo que diga si el nombre es femenino o

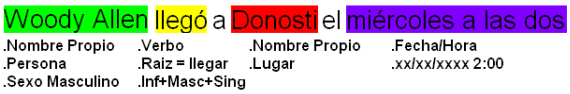

Figura 2.1: Ejemplo de Anotaciones 1

masculino. El hecho de permitir que las anotaciones tengan atributos permite representar otros tipos de análisis más complejos ya que una anotación puede tener a otras anotaciones como atributos.

Por ejemplo, el análisis de co-referencias de un texto identifica los antecedentes (unidades cuya semántica es un elemento del mundo exterior), los referentes (unidades cuya semántica está determinada por el antecedente) y las relaciones entre éstos [13]. En la figura 2.2 se muestra el análisis de co-referencias para las oraciones: «El castillo en Camelot fue la residencia de Arturo hasta el 536. En ese año, el rey lo trasladó de allí a Londres, y con él a toda su corte». Se muestran los antecedentes, los referentes, y las relaciones entre éstos.

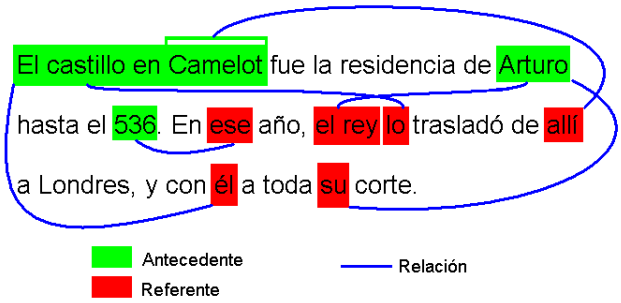

Figura 2.2: Ejemplo de Anotaciones 2

Como puede verse en la figura, es difícil visualizar este tipo de análisis donde se tienen anotaciones relacionadas entre sí, incluso en este ejemplo se tienen anotaciones solapadas: «El castillo de Camelot» y «Camelot». En el capítulo 4 se trata con más detalle el tema de la visualización de resultados.

2.2. Sistemas de PLN

El objetivo de los sistemas de PLN es permitir analizar lenguaje natural mediante la utilización de módulos o herramientas que, en general, se usan encadenadas. Dado un conjunto de módulos, los sis-

temas permiten al usuario ordenarlos de cierta forma para formar un flujo de componentes. Este conjunto de módulos en cadena, realiza un análisis para alguna aplicación de PLN.

Algunos tipos de análisis son utilizados por la gran mayoría de las aplicaciones de PLN. Por ejemplo, un análisis muy común consiste en la *tokenización* de texto (identificar los *tokens* individuales, como palabras y signos de puntuación). Este análisis es utilizado como primer paso en muchos análisis de las diferentes aplicaciones de PLN.

En la actualidad, se han desarrollado varios sistemas de PLN o plataformas que permiten construir aplicaciones a partir de la combinación de componentes de análisis. En este proyecto, se estudiaron varios sistemas; se estudió la arquitectura de organización, el formato de representación, la visualización de resultados de análisis, etc.

En primer lugar, se estudiaron algunas propuestas de arquitecturas. Entre ellas se tiene la arquitectura TIPSTER [19], desarrollada desde 1994, y la arquitectura ATLAS [2], cuyo desarrolló se centró principalmente en abstraer la gran diversidad de anotaciones lingüísticas que puede existir.

Luego, se estudiaron sistemas de PLN que en el algunos casos son implementaciones de estas arquitecturas. Un ejemplo es la plataforma GATE [10], desarrollada en la Universidad de Sheffield desde 1995. Esta plataforma ha sido usada en una amplia variedad de proyectos de investigación y desarrollo del área de PLN. GATE implementa la arquitectura mediante un *framework* y además provee un entorno de desarrollo construido sobre éste. Por otro lado, se tiene la plataforma Ellogon [8], desarrollada desde 1998. Ellogon es una plataforma que ofrece un conjunto de facilidades, que incluyen herramientas de procesamiento y visualización de resultados, soporte para utilización de recursos léxicos, etc. Además, se estudiaron sistemas más recientes como LinguaStream [17], en continuo desarrollo desde el año 2001, y UIMA [14] desarrollada por IBM en los últimos años.

Arquitecturas de Organización

Los sistemas de PLN estudiados tienen arquitecturas similares. Cada plataforma permite encadenar un conjunto de módulos de análisis de acuerdo a una jerarquía de ejecución. En general, los sistemas permiten crear flujos lineales de módulos, es decir, una especie de «tubería» de módulos donde las salidas de uno son las entradas del que le sigue.

También existen arquitecturas que permiten otro tipo de flujos más complejos, como un grafo de componentes. Este es el caso de

la arquitectura del sistema LinguaStream, la cual además de permitir encadenar los componentes de análisis como lo hacen las demás, permite que los componentes tengan varios puntos de entrada y/o salida, formando así un grafo acíclico dirigido de componentes. Esto es muy importante ya que los distintos puntos de salida de un componente pueden conectarse a cadenas (o grafos) diferentes, cuyos resultados podrán a su vez fusionarse en un componente que aceptará varios puntos de entrada. Para formar la cadena, LinguaStream provee un editor gráfico que permite elegir módulos de una «paleta» y conectarlos entre sí mediante tuberías. En la figura 2.3 puede verse un ejemplo de este editor.

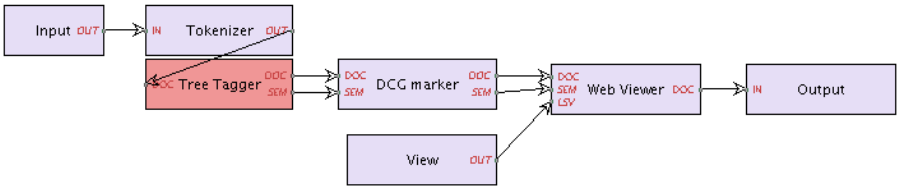

Figura 2.3: Cadena de Tratamiento de LinguaStream

Por otra parte, la implementación actual de la plataforma UIMA soporta flujos lineales entre componentes con ramificaciones condicionales basadas en el lenguaje del documento. Sin embargo, el motor del flujo puede ser fácilmente adaptado para soportar especificaciones de flujos más complejos.

En principio, todas las arquitecturas enfocan la manipulación de textos como módulos que tienen como entrada textos y producen como salida los textos iniciales más cierta información resultado del análisis. Por ejemplo, la arquitectura TIPSTER creada desde 1994 y pensada principalmente para aplicaciones de recuperación y extracción de información. Luego de que se diseñó TIPSTER, se realizaron implementaciones de esta arquitectura, con algunas modificaciones. Ejemplos de estas implementaciones son Ellogon y GATE.

Por otro lado, existen arquitecturas que no solamente están enfocadas a manipulación de textos, sino también a otros formatos como ser audio o video. Este es el caso de la arquitectura ATLAS, que además de tener en cuenta texto también tiene en cuenta otro tipo de señales como audio. La arquitectura UIMA, no solo tiene en cuenta texto y audio, sino también video.

En cuanto a los módulos que las arquitecturas permiten encadenar, generalmente se pueden dividir en varios tipos. Por ejemplo, la arquitectura GATE sugiere que se pueden dividir en tres clases de

recursos: Recursos del Lenguaje (representa entidades como léxicos, ontologías, etc.), Recursos de Procesamiento (representa entidades principalmente algorítmicas, como analizadores, etiquetadores, etc.) y Recursos Visuales. Es importante destacar que un usuario puede implementar e integrar a la plataforma nuevos recursos, de cualquiera de los tres tipos descritos anteriormente.

Por otra parte, la arquitectura de la plataforma UIMA, consta de bloques básicos llamados motores de análisis o AEs (*Analysis Engines*). Estos AEs representan los módulos de análisis que se componen para analizar un documento e inferir atributos descriptivos. Pueden existir distintos tipos de AEs, por ejemplo, un AE para recuperar una colección de documentos de una base de datos, otro AE para analizar texto, etc.

Módulos de Análisis

Las plataformas permiten que el usuario integre nuevos módulos de análisis. En algunos sistemas esta tarea es más fácil que en otros. Además, existen diferencias entre los lenguajes de programación que soportan. El lenguaje elegido por cada plataforma es variado; los componentes suelen implementarse en Java, C/C++, Tcl/Tk, Phyton, etc. De todas formas, el lenguaje favorito para la implementación de la plataforma es Java, seguramente debido a su portabilidad en los diferentes sistemas operativos.

Los módulos de análisis suelen utilizar recursos, por ejemplo, diccionarios o gramáticas. Además, cada módulo tiene un conjunto de parámetros de entrada. La asignación de valores para estos parámetros puede verse como una «configuración» que se realiza antes de utilizarlo para analizar. Un ejemplo de parámetro de entrada es el lenguaje del texto a analizar. Generalmente los recursos que los módulos utilizan, son parámetros de entrada que apuntan al archivo asociado al recurso.

En general, los sistemas cuentan con un conjunto de módulos de análisis integrados. En algunos casos, estos módulos realizan los análisis más comunes que utilizan la mayoría de las aplicaciones de PLN. Entre ellos se incluye: *tokenizador*, analizador morfológico, separador de oraciones, etc. Por ejemplo, la plataforma GATE provee un conjunto de recursos que constituyen un sistema denominado ANNIE *(A Nearly-New Information Extraction System)*. ANNIE fue desarrollado en el contexto de extracción de información pero cuenta con módulos que realizan las tareas más comunes de procesamiento.

Formato de Representación

Otro aspecto importante a tener en cuenta es la forma en que los componentes comparten la información del análisis. En general, las plataformas definen un formato o estructura en la cual cada módulo de la cadena puede agregar sus resultados de análisis. Estas estructuras son bastantes similares entre las plataformas, pero el hecho de que sean diferentes y no sean compatibles entre sí es un problema.

Todas las plataformas estudiadas utilizan una estructura de marcas o anotaciones para representar el análisis. En general, pueden haber muchos tipos de anotaciones, entonces, es importante definir una estructura que abarque todos los tipos posibles. El desarrollo de la arquitectura ATLAS se enfocó principalmente a proveer una abstracción sobre la diversidad de anotaciones lingüísticas. La idea principal fue enfocarse en la generalidad y extensibilidad y no tener en cuenta sólo texto, sino también otro tipo de señales como audio.

El formato de representación que utilizan las plataformas permite realizar anotaciones. Estas anotaciones tienen un índice de inicio y fin. Si se está analizando texto, los índices de inicio y fin pueden ser punteros a caracteres. Una particularidad de LinguaStream es que permite definir la granularidad de análisis para cada componente, por ejemplo: carácter, palabra, oración, etc. En este caso, los índices de inicio y fin dependen de la granularidad definida.

En general, las anotaciones pueden tener una serie de atributos. Estos atributos pueden ser de diferentes tipos, y pueden llegar a ser punteros a otras anotaciones, lo que permite formar un grafo de anotaciones para representar el análisis.

El formato de representación de texto que utiliza GATE es una modificación de TIPSTER y además es compatible con ATLAS. Un documento en GATE puede tener una o más capas de anotaciones: una capa anónima (por defecto) y tantas capas con nombre como sean necesarias. Una capa de anotaciones es organizada como un grafo dirigido acíclico en el cual los nodos son lugares particulares del contenido del documento y los arcos son anotaciones que van desde el lugar indicado por el nodo inicio del arco hasta el lugar que apunta el nodo final del arco.

Al igual que en GATE, en LinguaStream las anotaciones son organizadas en capas. Cada componente de análisis produce su propio marcado, basándose en los análisis anteriores, y los marcados resultantes se organizan en capas independientes y jerarquizadas. Las capas se pueden solapar, superponer y enlazar.

Una característica importante de UIMA es que puede trabajar

con varios formatos como ser audio, video, texto. Por ejemplo, si se tiene una conversación en formato audio y en formato texto, la conversación se puede analizar utilizando ambos formatos a la vez. Esto hace que que la plataforma se distinga entre las demás. UIMA define una estructura de análisis común *(CAS Common Analysis Structure)* mediante la cual los Anotadores comparten la información del análisis. CAS es una estructura de datos basada en objetos la cual permite la representación de objetos, propiedades y valores.

Ambiente y Visualización de Resultados

El ambiente que proveen los sistemas de PLN es un aspecto muy importante a tener en cuenta. Como se mencionó anteriormente, existen personas que no son del área informática, como los lingüistas, que necesitan hacer uso de estos sistemas. La plataforma UIMA no cuenta con una interfaz de usuario propia, su ambiente se ejecuta como *plugin* del entorno de desarrollo Eclipse [18]. Esto es un problema para usuarios que no son del área informática. Este tipo de usuarios no necesitan saber cómo integrar módulos pero sí necesitan saber cómo utilizarlos para poder analizar lenguaje natural. Al no estar familiarizados con el entorno Eclipse, esta tarea puede ser bastante difícil.

Por otro lado, la plataforma LinguaStream provee un ambiente orientado al usuario. En este ambiente es fácil para un usuario seleccionar un conjunto de componentes y formar una cadena para analizar un texto. Esta tarea se hace gráficamente utilizando un editor de cadenas como el que se ve en la figura 2.3. Teniendo la plataforma instalada, el proceso de análisis se realiza de forma muy intuitiva, incluso sin necesidad de estudiar el manual de usuario.

En general, todas las plataformas tienen un método de visualización «crudo» el cual muestra las anotaciones en el texto marcadas en diferentes colores y seleccionando una de ellas se muestran los valores de sus atributos. Esta forma de visualizar anotaciones puede ser útil en muchos casos pero en otros, por ejemplo para visualizar un árbol sintáctico, puede que no. En este aspecto se destaca la plataforma Ellogon que posee varias formas de visualizar los resultados. Por ejemplo, consta de una vista especial para árboles sintácticos que muestra la representación gráfica de un conjunto de anotaciones que describen una estructura arborescente. En la figura 2.4 se muestra un ejemplo.

En GATE se puede implementar todo tipo de recursos, y un tipo son los recursos visuales. Esto permite implementar nuevas «ventanas» especificando lo que se mostrará en cada panel. De todas

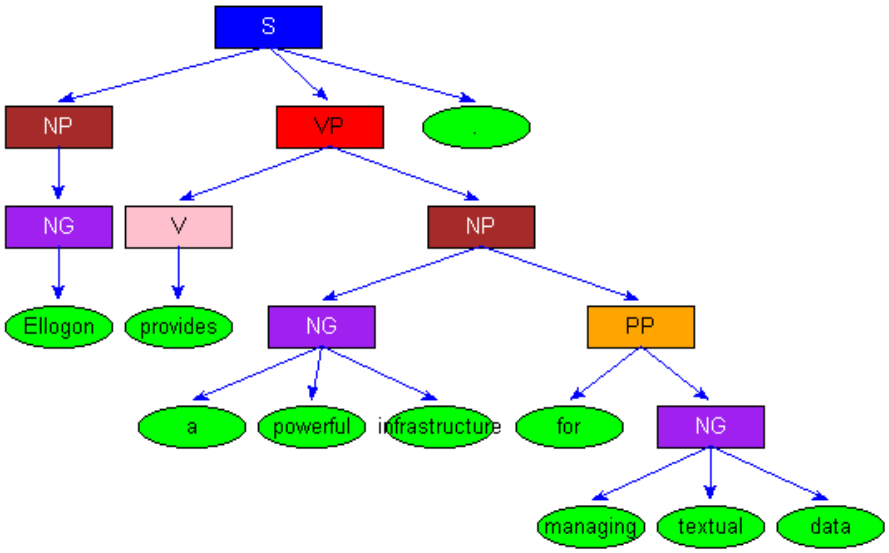

Figura 2.4: Árbol Sintáctico en Ellogon

formas, no existe ningún componente implementado para visualizar anotaciones de otra forma que no sea la mencionada. Por otra parte, LinguaStream provee varios componentes destinados a esta tarea, los cuales pueden colocarse en la cadena formada por los módulos de análisis.

Compatibilidad entre Plataformas

Los sistemas de PLN, en general, consisten en una plataforma, y como tal, no realizan ningún procesamiento lingüístico. Los módulos de análisis que se integran a la plataforma son los que se encargan de realizar el procesamiento. Por lo tanto, es importante que las plataformas sean compatibles entre sí. Esto permite que los usuarios desarrollen módulos de análisis en la plataforma de su preferencia y los puedan compartir de forma que los utilicen otros usuarios en conjunto con sus propios módulos en otra plataforma. Ellogon ofrece compatibilidad con la plataforma GATE, permitiendo transferir fácilmente componentes desde GATE hacia Ellogon. Además, se pueden exportar e importar resultados desde o hacia el formato que utiliza GATE.

Luego de la aparición de UIMA, otras plataformas desarrolladas un tiempo atrás han agregado componentes o funciones para permitir su interoperabilidad. La arquitectura de UIMA tiene muchas semejanzas con la arquitectura de GATE. Por esta razón, GATE

provee una capa de interoperabilidad para poder incluir componentes de UIMA en aplicaciones de GATE y viceversa, permitiendo que GATE utilice las ventajas del desarrollo flexible que posee UIMA y, a su vez, que los usuarios de UIMA puedan acceder a la gran variedad de recursos que GATE ya tiene disponibles.

2.3. Conclusiones de la Investigación

En el estudio realizado se encontraron muchas características en común entre los sistemas de PLN estudiados. Las plataformas manejan una serie de conceptos comunes a todas, pero cada una le asigna un nombre diferente: un módulo que realiza análisis sobre texto, en GATE se denomina recurso de procesamiento, en LinguaStream modelo de análisis y en UIMA motor de análisis (AE). Básicamente, todas las plataformas manejan un conjunto de módulos que realizan cierto procesamiento sobre el texto, estos módulos se pueden conectar y formar una tubería (o grafo en algunos casos) para que el resultado de análisis de un módulo sea la entrada del siguiente. Los módulos siempre tienen parámetros de entrada, los cuales pueden ser archivos que contienen diccionarios, gramáticas, etc. La forma de representar los resultados de análisis de los módulos varía de una plataforma a otra pero en general siempre se tienen anotaciones y cada una tiene un conjunto de atributos como ser las posiciones de inicio y fin, etc.

Las plataformas tienen muchas similaridades entre sí pero también tienen diferencias que hacen que algunas sean más fáciles de utilizar, más flexibles, extensibles, etc.

La elección de la plataforma a utilizar depende mucho de lo que se quiera hacer con ella, pero también hay ciertos aspectos importantes como ser la curva de aprendizaje que conlleva. Siempre están los usuarios que prefieren escribir comandos en una consola de UNIX a tener una ventana de Microsoft Windows. Para el caso de las plataformas también se puede llegar a hacer una comparación de ese estilo, ya que hay usuarios que pueden preferir plataformas como LinguaStream o Ellogon —las cuales poseen interfaces gráficas para la creación de aplicaciones donde se pueden crear diagramas de componentes al estilo UML— o plataformas como UIMA o GATE, donde la interfaz no es tan intuitiva y lo que se tiene es una lista de nombres de componentes a los que se les asigna un orden.

Por otra parte, en el estudio realizado pudo verse que existen algunas carencias en cuanto a la visualización de resultados de análisis. Algunos sistemas, como Ellogon, proveen métodos de visualización

más avanzados que el clásico en que se muestran las anotaciones con un color de fondo diferente para cada una. Éstos no son muy genéricos ya que están pensados para algún tipo de análisis en especial, como el visualizador de árboles sintácticos. Existe la necesidad de contar con una forma de visualización que permita cubrir los distintos tipos de análisis y que además tome en cuenta los problemas que surgen al mostrar anotaciones, como el solapamiento.

Dados los resultados obtenidos en la investigación, se concluyó que no era razonable desarrollar por completo una nueva plataforma de PLN. Esto se debe a que en la actualidad ya existen varios sistemas de ese tipo. Por otro lado, se vio que a pesar de que existen varias plataformas para PLN, no existe ninguna con un ambiente web, todas son aplicaciones de escritorio; incluso UIMA no tiene una interfaz de usuario propia.

El hecho de tener un ambiente web tiene algunas ventajas. En primer lugar, el usuario no necesita instalar la aplicación en un equipo, la misma está disponible simplemente accediendo a la red. Por otro lado, los usuarios pueden compartir sus módulos de análisis y utilizarlos en conjunto con otros.

La idea consiste en desarrollar una plataforma para PLN pero no por completo, esto es, utilizando alguno de los sistemas estudiados como «motor». En la investigación realizada se vio que, en general, todos los sistemas tienen arquitecturas similares. El objetivo es utilizar uno de estos sistemas existentes como un núcleo para la plataforma, desarrollando componentes que trabajen en conjunto con éste.

Por otra parte, dado que se decidió no desarrollar por completo la plataforma, se propuso hacer más énfasis en las características de la interfaz de usuario, principalmente en la visualización de resultados. Se propone implementar una visualización de resultados de análisis que tome en cuenta problemas como el solapamiento de anotaciones y sea lo más entendible posible por parte del usuario.

Entre los sistemas estudiados, se seleccionó la plataforma UIMA para utilizar como núcleo. Los motivos de esta elección fueron varios. En primer lugar, la arquitectura UIMA cumple con los requerimientos más importantes que se tenían en cuanto a los módulos de análisis y al estándar para la representación del texto. Además, UIMA es un sistema que se puede usar fácilmente debido a que tiene una API con toda la información que se necesita en cuanto a clases y procedimientos.

El proyecto entonces consiste en el desarrollo de una plataforma para PLN que utiliza UIMA como núcleo y tiene un ambiente web en el que se permite al usuario, entre otras cosas, integrar sus mó-

dulos de análisis y utilizarlos encadenados con otros existentes para analizar texto y visualizar el resultado.

Capítulo 3

Lavinia [1]

Dentro de los objetivos del proyecto, se tenía el desarrollo de una plataforma amigable y de fácil uso para la selección y ordenamiento de herramientas a aplicar sobre texto en lenguaje natural. Como se mencionó en el capítulo anterior, este objetivo se vio modificado al realizar la investigación del estado del arte. Dado que ya existen plataformas para PLN, se decidió que no era razonable desarrollar una por completo.

El objetivo del proyecto pasó a ser el desarrollo de Lavinia, un ambiente para PLN orientado al usuario. La propuesta consiste en el desarrollo de una plataforma que utiliza UIMA como núcleo, agregándole los componentes necesarios para tener como resultado final un ambiente web para PLN donde el usuario pueda, entre otras cosas, integrar sus propios módulos de análisis y utilizarlos encadenados para analizar texto y visualizar los resultados.

Se propone el desarrollo de un ambiente web de fácil uso, de manera de que pueda ser utilizado por distintos tipos de usuario, no necesariamente del área informática. El ambiente web consiste en la interfaz de usuario de la plataforma. Mediante este ambiente, los usuarios deben poder integrar módulos de análisis a la plataforma, manejar los recursos que utilizan los componentes (diccionarios, gramáticas, etc.), utilizar un conjunto de módulos en cadena para analizar un texto o un conjunto de textos (corpus). Además, se propone el desarrollo de otras funcionalidades, como la exportación de resultados de análisis que pueden ser de mucha utilidad en el desarrollo de aplicaciones de PLN. Por otra parte, se propone la implementación de una visualización de resultados intuitiva y genérica desde el punto de vista que sirva para representar una gran variedad de tipos de análisis. Se pretende que la visualización de resultados

[1] Lavinia era hija de Latino, el rey de los latinos. Es considerada héroe en la mitología griega.

sea innovadora, y que resuelva problemas como el solapamiento de anotaciones.

El objetivo de este capítulo es realizar una descripción de Lavinia. En la primer sección, se presentan las funcionalidades que la plataforma provee a través de su ambiente web. Luego, en la segunda sección, se describen los componentes que han sido integrados a Lavinia.

3.1. Funcionalidades

En cuanto a funcionalidades provistas por la aplicación, un usuario desde la página web puede agregar un nuevo componente a la plataforma. Para integrar el componente debe proporcionar al sistema todos los datos del mismo, esto incluye parámetros, tipos de entrada, tipos de salida, recursos que utiliza, y además debe adjuntar un archivo JAR con la implementación en UIMA del componente.

El hecho de permitir integrar componentes vía web tiene la ventaja de que cualquier usuario de la plataforma puede desarrollar un componente, integrarlo, y probarlo en conjunto con otros ya integrados. Por otro lado, tiene una desventaja importante: pueden existir usuarios malintencionados que dentro de la implementación del componente incluyan código que pueda dañar el sistema y el computador en que se ejecuta.

Lo que se hizo fue liberar una versión de Lavinia reducida, es decir, con menos funcionalidades. En esta versión denominada «Lavinia Internet», el usuario puede ver los detalles de los componentes integrados y realizar análisis. Se eliminaron las funcionalidades que permiten modificar la información que la plataforma mantiene sobre componentes y recursos. Esta versión está pensada para ser publicada en la web.

Además de poder integrar un nuevo componente, los usuarios pueden modificar la información de los existentes, esto incluye parámetros, sistema de tipos, recursos. Se permite también eliminar un componente existente, aunque esto debe hacerse con cuidado ya que el mismo podría tener como salida algún tipo de anotación que otro componente necesite como entrada y este último podría quedar inutilizable. Las funcionalidades de modificar y eliminar componentes también deberían ser privilegiadas y accedidas únicamente por parte de usuarios administradores.

La funcionalidad más importante provista por el ambiente es la de análisis de texto o corpus de texto a través de la página web. Para realizar el análisis el usuario sigue varios pasos. En primer lugar, el

usuario determina el flujo de componentes, para esto, selecciona un conjunto de componentes existentes en la plataforma y les asigna un orden de ejecución. En la figura 3.1 se muestra parte de la página web.

Figura 3.1: Selección de Componentes

En la segunda etapa, el usuario ingresa o selecciona los valores de los parámetros para todos los componentes del flujo. Luego, si los valores ingresados son válidos, el usuario puede analizar el texto. En la figura 3.2 se muestra parte de la página web donde se configuran los parámetros, en el ejemplo, se están configurando los parámetros del componente *POSTagger*.

En el tercer paso, el usuario ingresa el texto a analizar o selecciona los archivos del corpus, y los tipos de anotaciones que se quieren ver asignándoles un orden de prioridad. En el siguiente capítulo se explican los detalles de la visualización de resultados de análisis.

Luego de analizar el texto, el usuario en la página web puede ver el resultado del análisis con la configuración de tipos de anotaciones y colores que seleccionó, pudiendo modificarla a su gusto.

En las figuras 3.3 y 3.4 pueden verse partes de la página web del tercer paso. En la primer figura se muestran los nombres de los archivos del corpus que se analizó y el resultado de uno de ellos. En la segunda, se muestra la configuración de tipos de anotaciones y colores seleccionada y el resultado del análisis.

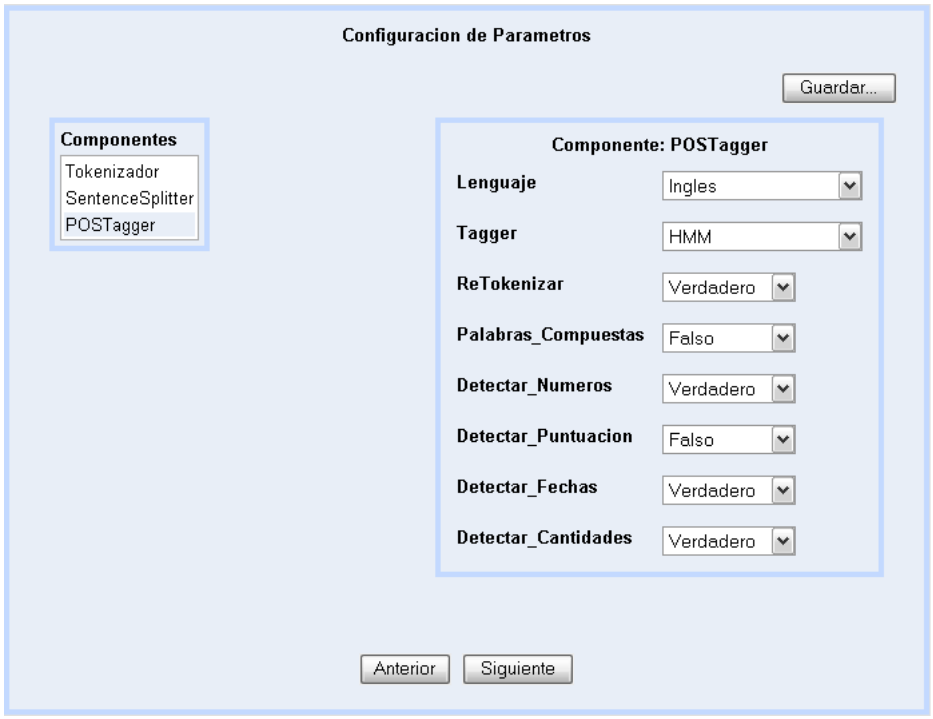

Figura 3.2: Configuración de Parámetros

El ambiente permite también poder guardar y cargar un flujo de componentes y sus parámetros. Cuando se quiere realizar un análisis, la selección del flujo de componentes y configuración de parámetros puede ser una tarea tediosa, por ese motivo, esta funcionalidad permite que un usuario guarde en un archivo el flujo y los parámetros de manera de poder utilizarlos en otro momento. Estas funcionalidades son importantes, no solo desde el punto de vista de lo que permiten hacer, sino también desde el punto de vista de la experiencia del usuario ya que se comprobó que mejoran la interacción con el ambiente al intentar en cierta forma «ayudarlo».

Otra de las funcionalidades provistas por el ambiente es la de guardar los resultados de análisis. El formato en el que se guardan los resultados es XMI *(XML Metadata Interchange)*. El hecho de permitir guardar los resultados de análisis en un archivo, en principio puede servir para abrirlos en otro momento, por ejemplo, utilizando el *plugin* de UIMA para Eclipse.

Desde el ambiente web se puede realizar la configuración de los recursos externos que utilizan los componentes. Se pueden agregar nuevos recursos, modificar y eliminar los existentes. Esta funciona-

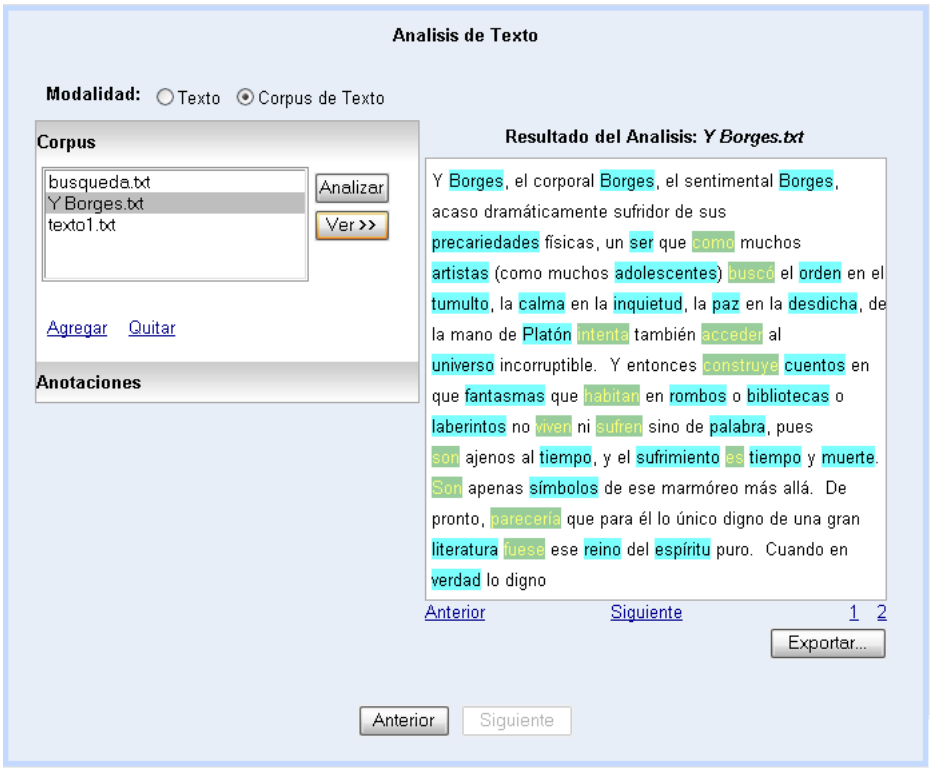

Figura 3.3: Análisis de Corpus

lidad hace que sea más amigable la configuración de recursos ya que se hace todo en la interfaz gráfica, sin embargo, en general, siempre va a utilizarse desde el computador donde se ejecuta el servidor ya que las rutas son relativas a su sistema de archivos.

3.2. Componentes

La distribución de Lavinia tiene un conjunto de módulos de análisis integrados. Es importante destacar que Lavinia es una plataforma. Se enfatiza la palabra «plataforma» ya que Lavinia no pretende realizar ningún tipo de procesamiento lingüístico. Esto se hace mediante el uso de módulos de análisis integrados. Por lo tanto, Lavinia no necesita de los módulos de análisis integrados para funcionar, en realidad es a la inversa, los módulos funcionan y se pueden encadenar gracias a Lavinia.

Para cumplir con el objetivo del proyecto de contar con módulos que realicen las principales tareas de PLN, se integraron módulos

que encapsulan parte del conjunto de herramientas FreeLing [9].

El primer módulo que se integró fue el *Tokenizador*. Este módulo se encarga de detectar 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). Este módulo no requiere ningún tipo de anotación como entrada.

En segundo lugar, se tiene un módulo que detecta las oraciones del texto, éste se denomina *SentenceSplitter*. 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.

Se tiene además un módulo que 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.

Además, se integró un módulo que no entraba dentro del alcance del proyecto. El módulo fue desarrollado en el marco del curso Introducción al Procesamiento de Lenguaje Natural [15]. El análisis que realiza consiste en la desambigüación semántica de un conjunto de palabras predefinido (se ingresan como valores para un parámetro). Para cada palabra de la lista, se asigna el sentido más probable entre los sentidos de la base de datos léxica *WordNet* [4]. El módulo utiliza los resultados del *tokenizador*, el separador en oraciones, y el etiquetador morfológico. Además, utiliza *WordNet* y se conecta al buscador *Google* para obtener información.

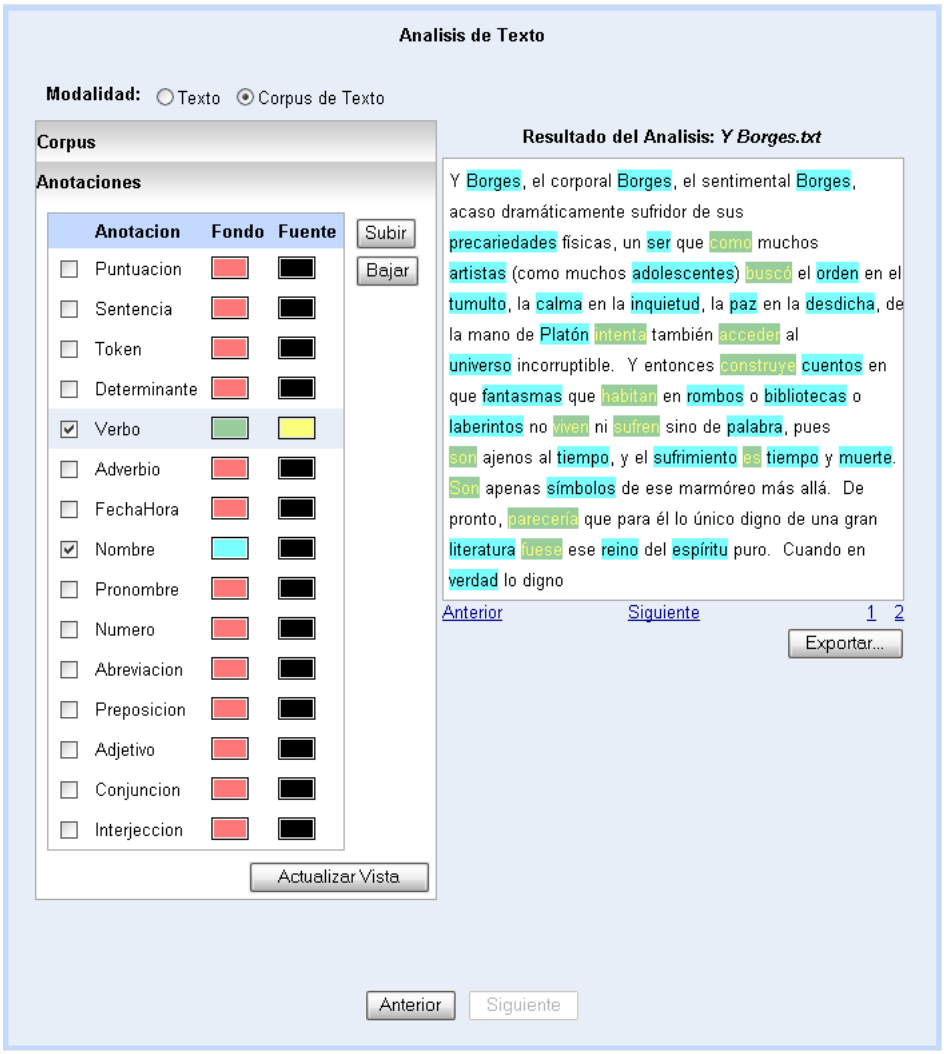

Figura 3.4: Análisis de Corpus, Tipos de Anotaciones

Capítulo 4

Diseño e Implementación

El objetivo de este capítulo es presentar el diseño e implementación de la aplicación desarrollada. En primer lugar, se describe la arquitectura y los componentes principales del sistema. Además, se describen las principales características de la arquitectura UIMA. Por último, se describe cómo se realizó la implementación, justificando las decisiones tomadas en cuanto a las tecnologías escogidas.

4.1. Diseño

El sistema Lavinia consiste en un ambiente para procesamiento de lenguaje natural. Lavinia se puede dividir en dos partes. Por un lado, se tiene la plataforma donde se integran módulos y se realiza el procesamiento; por otra parte, se tiene la interfaz de usuario de la plataforma.

La plataforma tiene a UIMA como núcleo, utilizando la definición de módulo de análisis y representación estándar de resultados que el subsistema define. Además, Lavinia permite que los módulos de análisis utilicen recursos como diccionarios o incluso de otro tipo como herramientas de procesamiento externas a la aplicación.

Lavinia provee un ambiente web para permitir al usuario analizar texto y organizar los módulos de análisis de la plataforma, entre otras cosas. El diseño del ambiente web está orientado a la facilidad de uso. Se diseñó un algoritmo para solucionar algunos de los problemas de la visualización de resultados, teniendo en cuenta los solapamientos de anotaciones.

En la figura 4.1 se muestra muy esquemáticamente la interacción de un usuario con el sistema. Básicamente, el usuario utiliza el sistema por medio de su ambiente web. Este ambiente que se ejecuta en un *browser* se comunica con un servidor web donde se encuen-

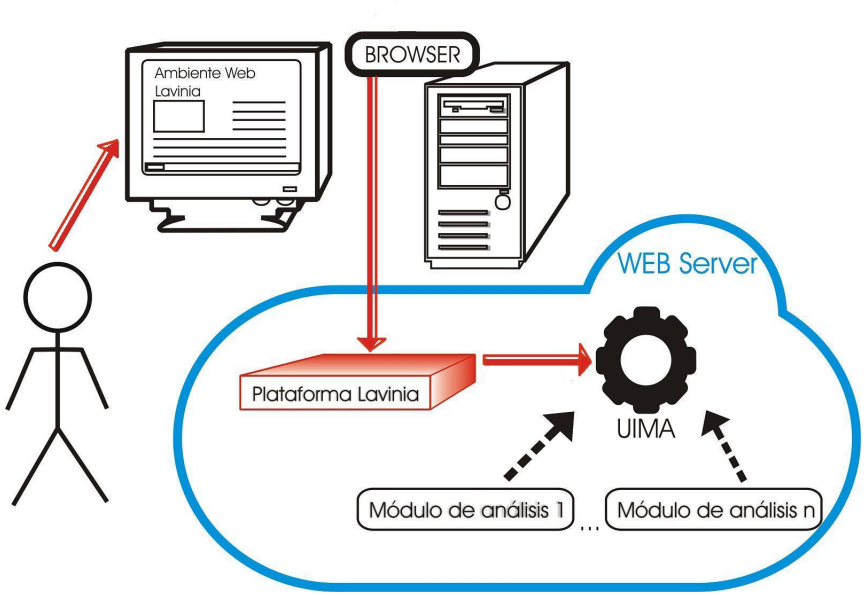

Figura 4.1: Esquema de interacción entre un usuario y Lavinia

tra instalada la plataforma Lavinia. La plataforma obtiene los datos ingresados por el usuario y la acción que éste requiere. En el caso típico, el usuario va a requerir el análisis de un texto o un conjunto de textos, para esto, selecciona un conjunto de módulos de análisis desde el ambiente, y les asigna un orden de ejecución. Estos módulos de análisis están implementados utilizando la arquitectura UIMA. De esta forma, Lavinia utiliza UIMA para realizar el análisis y luego de finalizado, envía al *browser* los resultados obtenidos para que sean desplegados en el ambiente web.

4.1.1. Arquitectura

La arquitectura del sistema sigue un modelo cliente-servidor. En la figura 4.2 se muestra un diagrama con los componentes principales y sus dependencias.

Servidor

En el servidor es donde se encuentra toda la lógica de la aplicación. Dentro del paquete «Lógica» del diagrama de la figura 4.2, se encuentran todos los servicios, que se ejecutan en el servidor como respuestas a pedidos que realizan los clientes.

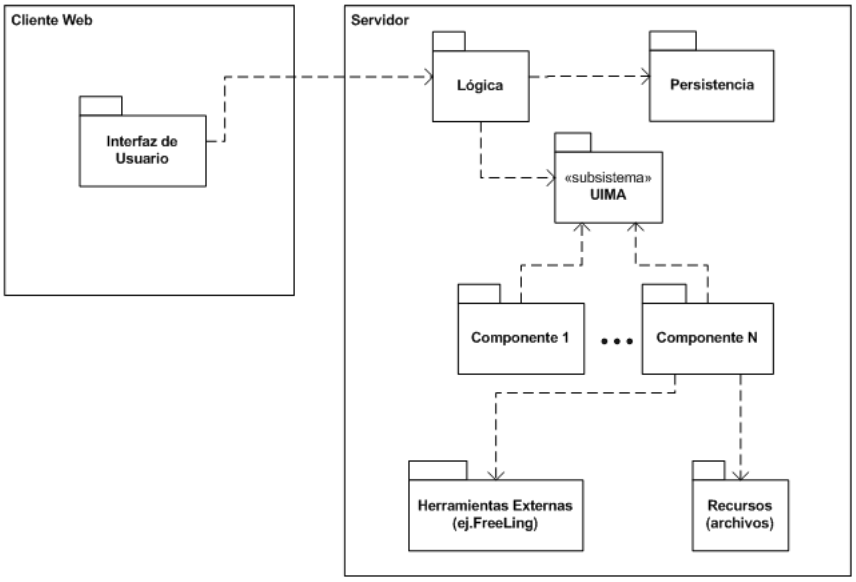

Figura 4.2: Arquitectura del Sistema

Lavinia depende del subsistema UIMA en varios aspectos. En primer lugar, se utiliza la definición de módulo de análisis de UIMA, es decir, cada módulo de Lavinia tiene un conjunto de parámetros para configurarlo, un sistema de tipos, y puede utilizar un conjunto de recursos y herramientas externas (ver sección 4.1.3). Además, se utilizan algunas funcionalidades que provee UIMA para exportar resultados de análisis a un archivo (serialización).

Los componentes o módulos de análisis, se implementan utilizando UIMA. Un programador puede implementar un módulo de análisis utilizando UIMA, por ejemplo, usando el *plugin* para Eclipse. Luego, el módulo se puede integrar a Lavinia. Es importante destacar que para implementar el módulo, no se necesita importar ninguna biblioteca de Lavinia, en otras palabras, no se crea ninguna dependencia entre los módulos de análisis y Lavinia.

La plataforma almacena un conjunto de datos en el módulo de «Persistencia». Básicamente, se guarda toda la información de los módulos de análisis integrados, los recursos que se tienen, y las herramientas externas que los módulos utilizan.

Cliente

El cliente es fino: básicamente se encarga de enviar y recibir datos desde y hacia el servidor.

Al diseñar la interfaz de usuario, se hizo especial énfasis en la facilidad de uso y en la satisfacción del usuario. Este último punto se puede ver favorecido si se logra cierta interactividad en la navegación web, es decir, cierta «dinámica». Para lograr esta dinámica, se requiere que la página web cambie en respuesta a diferentes contextos o condiciones. Existen dos formas de crear este tipo de interactividad [25]:

- Utilizando *scripting* en el cliente para cambiar el comportamiento de la interfaz en una página especifica, en respuesta a acciones del *mouse* o teclado o eventos de tiempo específicos.
- Utilizando *scripting* en el servidor para cambiar la página suministrada, entre páginas, adjustando la secuencia o recargando las páginas o el contenido.

El resultado de utilizar cualquiera de ambas técnicas se describe como página web dinámica.

En nuestro caso, para lograr la interactividad y dinámica que mejoran la satisfacción del usuario, se decidió utilizar páginas web dinámicas, diseñando el cliente para ser implementado con un lenguaje de *scripting* (por ejemplo, *JavaScript*).

4.1.2. Subsistema UIMA

El objetivo de esta sección es describir algunos detalles de la plataforma UIMA [14]. En la aplicación desarrollada no se utilizaron todas las características de UIMA, la idea es dar una descripción general del sistema y lo que se utilizó de éste .

En cuanto a la arquitectura, se tienen bloques básicos llamados motores de análisis (*AE, Analysis Engines*). Estos AEs representan los módulos de análisis que se componen para analizar un documento e inferir sus atributos descriptivos. Existe una tipo particular de AE que se denomina Anotador. Un Anotador es un módulo que contiene un algoritmo de análisis y realiza anotaciones sobre el texto.

Cada AE en UIMA tiene un sistema de tipos que define cuáles son los tipos que maneja el componente. Los tipos se pueden relacionar mediante herencia jerárquica y pueden tener atributos. Existe un tipo general utilizado en componentes de análisis del cual se pueden derivar tipos adicionales, este es el tipo *Anotación*. Este tipo es utilizado para anotar o etiquetar regiones. Un anotador, por ejemplo, puede crear una *Anotación* de tipo *Persona* para registrar que encontró un lugar donde se menciona una persona entre las posiciones i y j de un documento.

Una parte importante de la arquitectura de UIMA es cómo los anotadores representan y comparten los resultados. UIMA define una estructura de análisis común *(CAS, Common Analysis Structure)* con ese propósito. CAS es una estructura de datos basada en objetos la cual permite la representación de objetos, propiedades, y valores. Los tipos se organizan en una taxonomía y pueden tener atributos. Por ejemplo, el tipo Vehículo puede tener un atributo Marca, y Auto puede ser un subtipo de Vehículo, heredando ese atributo.

Los tipos tienen atributos, por ejemplo, Persona puede tener los atributos Edad y Ocupación. Los Sistemas de Tipos pueden organizarse en una taxonomía, por ejemplo, Compañía puede ser un subtipo de Organización.

Las anotaciones no son el único tipo que se puede encontrar en un CAS. Un CAS es un esquema de representación general y puede guardar un conjunto arbitrario de estructuras de datos para representar el análisis de documentos.Un ejemplo puede verse en la figura 4.3 adaptada de la documentación de UIMA [14].

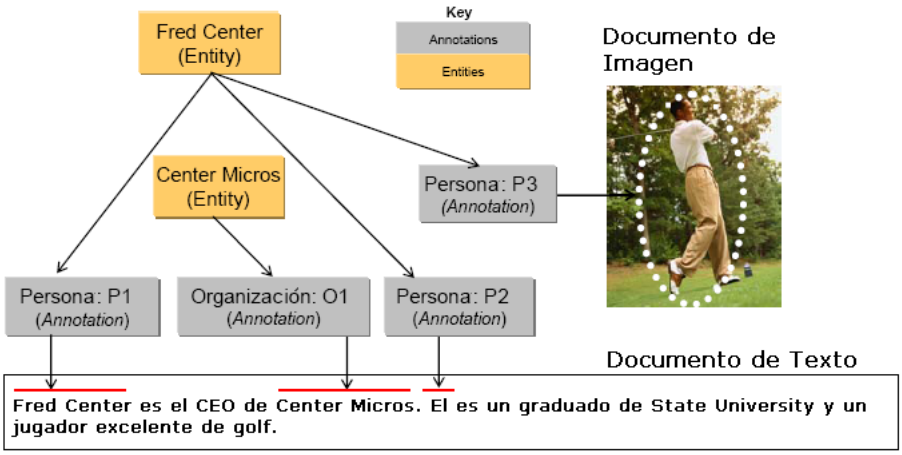

Figura 4.3: Objetos representados en un CAS

Para todos los componentes especificados en UIMA hay dos partes que se requieren para su implementación:

1. la parte declarativa y
2. la parte de código

La parte declarativa contiene datos que describen los componentes, su identidad, estructura, y comportamiento, esta parte se denomina

Descriptor del Componente. La parte del código que implementa el algoritmo puede ser un programa en Java.

Un descriptor de componente identifica el Sistema de Tipos que el componente utiliza, los tipos que requiere como entrada en un CAS y los tipos que planea producir como salida en un CAS.

Un AE simple o primitivo contiene un único Anotador. Sin embargo, los AEs se pueden organizar en flujos para formar otros AEs. Este análisis más complejo se denomina Motor de Análisis Compuesto (*AAE - Aggregate Analysis Engines*). Los usuarios que utilizan este tipo de AEs no necesitan saber como está construido internamente, sólo necesitan el nombre y sus entradas requeridas y tipos en la salida. Todo esto se encuentra en el descriptor del AE, el cual declara los componentes y la especificación del flujo. La especificación del flujo define el orden en que se deben ejecutar los componentes internos. En el diagrama de la figura 4.4 se muestran los conceptos mencionados [5].

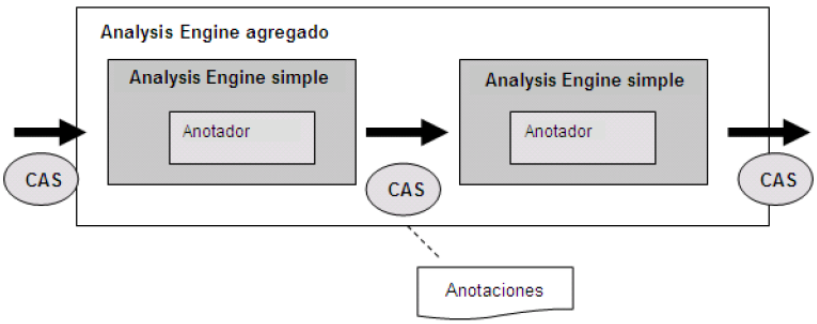

Figura 4.4: Actores principales de la arquitectura UIMA

4.1.3. Módulos de Análisis

Los módulos de análisis de Lavinia se implementan utilizando la arquitectura UIMA. El objetivo de esta sección es describir cómo están formados éstos módulos y cómo interviene Lavinia.

Cada módulo de análisis consta de un conjunto de datos. Además del nombre del módulo, se guarda el nombre del autor y una descripción del análisis que realiza.

Cada módulo tiene un conjunto de parámetros. Los parámetros de un componente sirven para configurarlo cada vez que se quiera usar para analizar texto. Se pueden ver como parámetros de entrada de un procedimiento. Por ejemplo, si un componente se encarga de identificar nombres de ciudades a partir de una lista predefini-

da, se podría tener un parámetro (multivaluado) con los nombres de las ciudades; estos valores se pueden modificar cada vez que el componente se utilice.

Existe la posibilidad de que un módulo tenga parámetros que no son multivaluados, es decir, toman un único valor, pero este valor lo puede elegir el usuario entre un conjunto que se define al integrar el módulo. Por ejemplo, puede existir un parámetro para definir el lenguaje del texto a analizar que si bien toma un solo valor, el valor se elige de una lista de opciones (español, inglés, italiano, etc.). Para realizar esta tarea, Lavinia guarda el conjunto de valores posibles que aparecerán en una lista de opciones cuando el módulo se quiera utilizar para analizar. Aquí nuevamente se intenta mejorar la experiencia del usuario facilitando la interacción con el sistema.

Otro aspecto importante que tiene un componente es el sistema de tipos, éste define las entradas que el componente necesita para realizar el análisis, y las salidas que va a generar. Por ejemplo, para realizar el análisis morfológico, necesitan conocerse los *tokens* del texto (las palabras individuales y símbolos de puntuación). Por lo tanto, este módulo va a necesitar como entrada el tipo *Token*, el cual será tipo de salida del módulo que realiza la *tokenización*.

Los tipos se ordenan en una estructura jerárquica: cada tipo tiene un «padre» y hereda sus atributos. El tipo del cual deben heredar las anotaciones es *Annotation*; éste contiene únicamente los atributos *inicio* y *fin* (índices de caracteres en el texto).

Por último, los componentes utilizan un conjunto de archivos, llamados recursos. Los recursos pueden ser externos a la aplicación (fuera de su estructura de directorios); en este caso se puede tratar de herramientas externas (es el caso de FreeLing). Los recursos también pueden ser internos a la aplicación y en este caso los archivos asociados se deben incorporar a la plataforma cuando se agrega el componente. Para esto, Lavinia mantiene información sobre los recursos existentes, permitiendo que un usuario pueda agregar, modificar o eliminar los mismos, de manera que puedan ser utilizados por los módulos para realizar el procesamiento.

4.1.4. Visualización de Resultados

Como se mencionó anteriormente, los resultados de análisis consisten en anotaciones en el texto, es decir, marcas que tienen un índice de inicio y fin. Además, cada anotación tiene una serie de atributos que, en particular, pueden referenciar a otras anotaciones del texto.

La visualización de este tipo de resultados es un problema difí-

cil de resolver, principalmente por tres motivos. En primer lugar, se puede tener cierto número de anotaciones solapándose. Si, por ejemplo, se marca cada anotación con un color diferente, y se tienen varias anotaciones de distinto tipo que coinciden (los índices de inicio y fin son el mismo), ¿de qué color será la marca? En segundo lugar, es difícil representar que una anotación haga referencia a otra en el texto. Por último, las tecnologías utilizadas para implementar la visualización pueden imponer restricciones sobre lo que se puede representar visualmente.

Al realizar el estudio del estado del arte para este proyecto, no se encontró una forma de visualización de resultados que solucione todos los problemas mencionados, en particular, el del solapamiento. Por tal motivo, se propone una solución que toma en cuenta los solapamientos de anotaciones.

En primer lugar se pensaron dos soluciones posibles. La primera, consiste en subrayar cada anotación y en caso de solapamiento, la línea se coloca por debajo de la que ya se dibujó. La segunda opción consiste en utilizar mezclas de colores para el color de fondo de las anotaciones. Por ejemplo, si una anotación que tiene color de fondo amarillo se debe solapa con una azul, la porción de texto que se solapa tendría color de fondo verde. Una forma de hacer esto es promediando los valores de *rgb* de cada anotación. El problema de esta solución es que no siempre es intuitivo ver qué colores participaron para formar determinada mezcla, y además ésta puede tender a colores oscuros si se solapan varias anotaciones.

La solución que se implementó fue considerada la mejor entre las opciones que habían, teniendo en cuenta las tecnologías que se utilizaron y la opinión de algunos usuarios. A continuación se describe la misma.

En primer lugar, la forma en que se visualizan las anotaciones depende de la prioridad que tiene cada tipo de anotación. Los tipos de anotaciones que son salidas del flujo de componentes, se ordenan en una lista, donde el tipo que está primero es el de mayor prioridad. Además, cada tipo de anotación tiene asociado un color que corresponde al color de fondo que se usará para marcarla en el texto. Es necesario destacar que el usuario puede seleccionar cuáles son los tipos de anotaciones que se van a mostrar, pudiendo omitir tipos para los cuales no necesita ver las marcas en el texto.

La tarea principal del algoritmo es marcar las anotaciones con el color de fondo que se le asignó al tipo, siguiendo el orden de prioridad. Se comienza por marcar en el texto las anotaciones del tipo con mayor prioridad, y se sigue por las anotaciones de los demás tipos (en orden). Cuando se debe marcar una región en el texto

que ya fue marcada (seguramente por una anotación con tipo de mayor prioridad), estamos ante un solapamiento. En este caso, lo que se hace es dibujar un borde alrededor de la región a marcar. Este borde será del mismo color que debería marcarse el fondo de la región. Por otra parte, si se quiere marcar una región del texto que ya fue marcada por otra anotación, es decir, ya tiene color de fondo, y además tiene un borde (son dos anotaciones solapadas), sólo se cambia el borde del primer y el último carácter. En este caso, otra opción sería no marcar nada, pero el hecho de marcar inicio y fin puede servir en muchos casos, un ejemplo de esto es el que explica a continuación.

Para entender mejor como funciona el algoritmo, veamos un ejemplo. En la figura 4.5 se tienen cuatro anotaciones:

1. «come», tipo=Verbo, inicio=8, fin=11
2. «El gato come pescado.», tipo=Oración, inicio=0, fin=20
3. «gato», tipo= Nombre, inicio=3, fin=6
4. «pescado», tipo=Nombre, inicio=13, fin=19

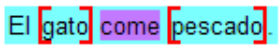

Figura 4.5: Ejemplo

El orden de prioridad para los tipos es el siguiente (de mayor a menor): Verbo (violeta), Oración (celeste), Nombre (rojo).

Cada carácter del texto se considera una *celda*. Inicialmente el texto no tiene ninguna marca. Se comienza por marcar la anotación que tiene la mayor prioridad, entonces, se cambia el fondo (violeta) de las celdas incluidas en «come».

La siguiente anotación a marcar es la de tipo Oración, que incluye todo el texto. Dado que ya tenemos una marcada, estamos ante un solapamiento. Lo que hacemos entonces es marcar el borde (celeste) alrededor de la región a marcar «El gato come pescado.», y luego como dentro de la región tenemos celdas que aún siguen sin color de fondo (todas las celdas menos «comen»), cambiamos el color de fondo de éstas (celeste).

Por último, tenemos las dos anotaciones de tipo «Nombre». Dado que en este caso, las regiones a marcar ya tienen color de fondo y

color de borde asignado, sólo marcamos el inicio y el fin de cada una (rojo).

El seudocódigo del algoritmo (simplificado) se muestra en el cuadro 2. Cada carácter del texto se considera una celda.

Cuadro 1 Seudocódigo algoritmo de visualización

```
Para cada tipo de anotación ti en orden de prioridad
    Para cada anotación aj del tipo ti
        Para cada celda entre aj.inicio y aj.fin
            Si la celda tiene fondo
                Si la celda tiene bordes
                    Sólo asignar borde a aj.inicio y aj.fin
                Sino
                    Asignar bordes
            Sino
                Asignar fondo
        Fin
    Fin
Fin
```

En las figuras 4.6 y 4.7 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 tipos de anotación. Además, en cada ejemplo se muestra el *popup* que aparece al hacer click sobre una celda, donde se muestran los valores de los atributos de todas las anotaciones de la celda.

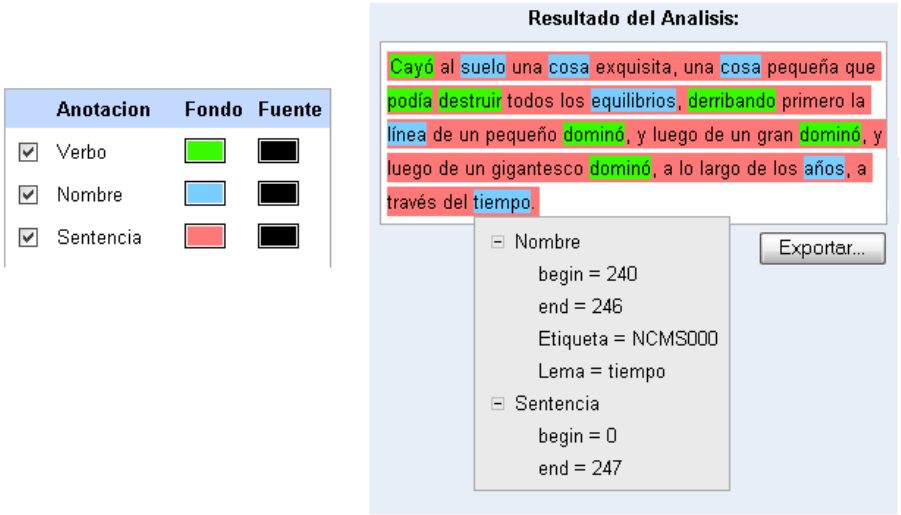

Figura 4.6: Ejemplo 1 de visualización de resultados

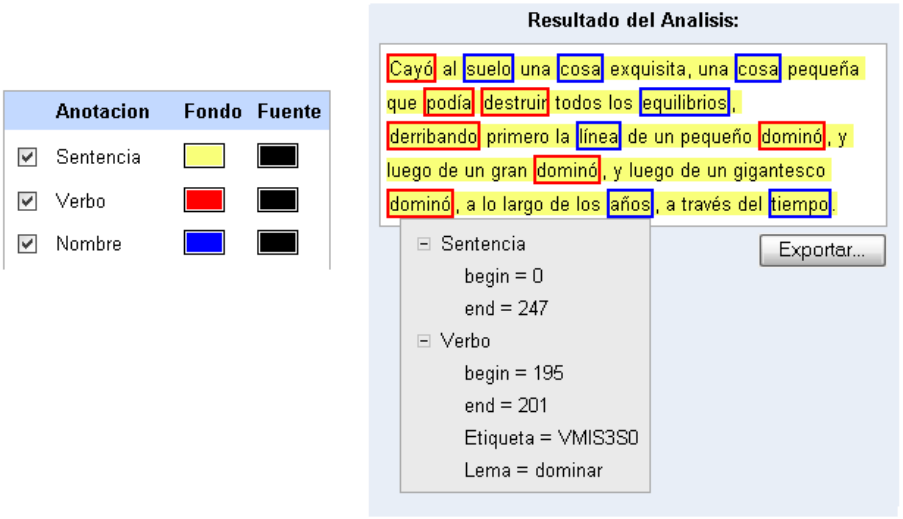

Figura 4.7: Ejemplo 2 de visualización de resultados

En la siguiente sección se explican los detalles de la implementación del algoritmo.

4.2. Implementación

4.2.1. Cliente

Como se mencionó en la sección anterior, el objetivo es utilizar un lenguaje de *scripting* para implementar la interfaz de usuario, por ejemplo Javascript. Este lenguaje es el que está más relacionado al concepto Web 2.0. Este concepto refiere a la segunda generación de páginas web, páginas dinámicas orientadas al usuario y haciendo especial énfasis en la interacción de éste con la aplicación.

La Web 2.0 es la representación de la evolución de las aplicaciones tradicionales hacia aplicaciones web enfocadas al usuario final. Se trata de aplicaciones que generen colaboración y de servicios que reemplacen las aplicaciones de escritorio [6].

Las siguientes técnicas son características de las webs construidas utilizando tecnología de la Web 2.0 [26]:

- CSS *(Cascade Style Sheets)* [22]
- Técnicas de aplicaciones ricas no intrusivas, como AJAX
- URLs sencillas y con significado

AJAX, acrónimo de *Asynchronous JavaScript And XML* (JavaScript y XML asíncronos), es una técnica de desarrollo web para crear aplicaciones interactivas. Éstas se ejecutan en el cliente, es decir, en el navegador de los usuarios, y mantienen comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad [24].

AJAX no constituye una tecnología en sí, sino que es un término que engloba a un grupo de éstas que trabajan conjuntamente, incorpora:

- presentación basada en estándares usando XHTML y CSS,
- exhibición e interacción dinámicas usando el *Document Object Model* (DOM),
- Intercambio y manipulación de datos usando XML and XSLT,
- Recuperación de datos asincrónica usando XMLHttpRequest,
- y JavaScript.

El modelo clásico de aplicaciones Web funciona de esta forma: La mayoría de las acciones del usuario en la interfaz disparan un requerimiento HTTP al servidor web. El servidor efectúa un proceso (recopila información, procesa números, comunicándose con varios sistemas propietarios), y le devuelve una página HTML al cliente. Este acercamiento tiene mucho sentido a nivel técnico, pero no lo tiene para una gran experiencia de usuario. Mientras el servidor esta haciendo sus tareas, el usuario está esperando. A la izquierda de la figura 4.8 se muestra el diagrama que representa esta interacción entre cliente y servidor.

Si estuviéramos diseñando la Web desde cero para aplicaciones, no querríamos hacer esperar a los usuarios. Una vez que la interfaz esta cargada, la interacción del usuario no debería detenerse cada vez que la aplicación necesita algo del servidor. De hecho, el usuario no tendría que darse cuenta que la aplicación se está comunicando con el servidor.

Una aplicación AJAX elimina la naturaleza de la interacción en la Web de «arrancar-frenar-arrancar-frenar» introduciendo un intermediario —un motor AJAX— entre el usuario y el servidor. Parecería que sumar una capa a la aplicación la haría menos reactiva, pero la verdad es lo contrario.

En vez de cargar un página Web, al inicio de la sesión, el navegador carga al motor AJAX (escrito en JavaScript). Este motor

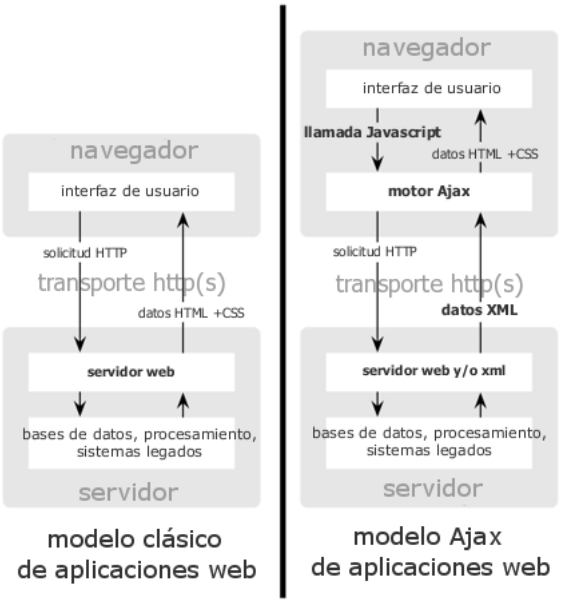

Figura 4.8: Modelo tradicional para las aplicaciones Web comparado con el modelo de AJAX

es el responsable por renderizar la interfaz que el usuario ve y por comunicarse con el servidor en nombre del usuario. El motor AJAX permite que la interacción del usuario con la aplicación suceda asincrónicamente (independientemente de la comunicación con el servidor). Así, el usuario nunca estará mirando una ventana en blanco del navegador y un icono de reloj de arena esperando a que el servidor haga algo. A la derecha de la figura 4.8 se muestra un esquema de esta interacción.

Cada acción de un usuario que normalmente generaría un requerimiento HTTP toma la forma de un llamado JavaScript al motor AJAX en vez de ese requerimiento. Cualquier respuesta a una acción del usuario que no requiera una viaje de vuelta al servidor (como una simple validación de datos, edición de datos en memoria, incluso algo de navegación) es manejado por su cuenta. Si el motor necesita algo del servidor para responder (sea enviando datos para procesar, cargar código adicional, o recuperando nuevos datos) hace esos pedidos asincrónicamente, usualmente usando XML, sin frenar la interacción del usuario con la aplicación [3].

Para desarrollar la aplicación utilizando el lenguaje JavaScript, hay varias opciones. Una de ellas es utilizar un *framework* como Prototype [20]. Este marco de trabajo permite programar en JavaScript utilizando Ajax fácilmente. Para este proyecto, se decidió no

programar directamente en JavaScript, sino utilizar un *framework* que permite programar en Java y se encarga de realizar la traducción a JavaScript y Ajax.

Se estudiaron dos marcos de trabajo, Google Web Toolkit (GWT) [11] y Echo2 de NextApp [1]. Estas herramientas permiten escribir aplicaciones web utilizando el lenguaje Java que luego es traducido por un compilador hacia código JavaScript. A la hora de elegir un marco de trabajo para trabajar hay que considerar varios aspectos. Lo mas importante es el tipo de aplicación que se quiere desarrollar. En nuestro caso en particular, la aplicación no presenta demasiada complejidad; la funcionalidad que podría verse afectada por el marco de trabajo seleccionado es la visualización de resultados. GWT es un marco de trabajo muy simple, es fácil de utilizar y cuenta con los elementos necesarios para desarrollar una aplicación relativamente sencilla. En contraste, Echo2 es un marco de trabajo muy avanzado, permite desarrollar aplicaciones más complejas y por lo tanto, es más difícil de utilizar.

Otra de las características importantes a considerar es el tiempo de respuesta. Para esto, se hicieron algunas pruebas utilizando ambas herramientas y pudo verse que Echo2 es bastante lento, comparado con GWT. Este es un punto muy importante a favor de GWT ya que aumenta la satisfacción del usuario al hacer más rápida y dinámica la interacción con el sistema. Además, se encontraron otras diferencias entre las herramientas: tanto Echo2 como GWT generan DHTML a partir de Java, la diferencia es que Echo2 utiliza un paso intermedio pasando de Java a *byte code* y a partir de éste genera DHTML, GWT compila Java directamente a DHTML.

Dadas las características mencionadas de cada marco de trabajo estudiado, se decidió optar por GWT para desarrollar la aplicación. En la siguiente sección se describe este marco de trabajo.

Google Web Toolkit es un *framework* de código abierto para crear aplicaciones Web 2.0 con tecnología AJAX. Este *framework* permite construir aplicaciones web dinámicas de forma sencilla usando Java como lenguaje de programación, que luego es compilado a JavaScript. GWT se encarga de producir código compatible con estándares y navegadores, el programador no necesita preocuparse por este aspecto. Luego de escribir el código de la aplicación en lenguaje Java, el compilador de GWT lo traduce a código JavaScript.

El hecho de programar la aplicación en lenguaje Java tiene varias ventajas. En primer lugar, permite a alguien con conocimientos de construcción de interfaces de usuario en Java realizar aplicaciones web AJAX, con esto se puede aprovechar la experiencia de muchos programadores que pueden usar sus practicas habituales de progra-

mación sobre la web. El código se puede ejecutar y depurar directamente sobre Java, aprovechando las posibilidades de depuración de entornos como Eclipse y aprovechando otras características como la completación automática, generación de código y refactorización.

Por otra parte, se pueden extender los componentes que el *framework* provee, y esto es tan sencillo como heredar de una clase Java. Para cosas más complejas se puede recurrir a JSNI (un interfaz que proporcionan para realizar llamadas nativas a código JavaScript). Esto fomenta la creación de bibliotecas de componentes para GWT, las mismas se pueden distribuir en archivos JAR.

El *framework* permite utilizar hojas de estilo (CSS) para poder separar la parte visual que tiene que ver con tipos de letra, colores, tamaños, fondos, de la parte de programación. Modificando un único archivo, sin necesidad de re-compilar el código, se puede cambiar todo el aspecto de la aplicación.

Además de las ventajas mencionadas, a causa de que el *framework* es bastante reciente, se tienen algunas desventajas. Actualmente, la versión de GWT disponible es Beta, por lo tanto, contiene errores y problemas que aún no han sido corregidos. Por otra parte, existen algunas carencias que podrían hacer más fácil la programación utilizando el *framework*. No se cuenta con un editor visual para crear las páginas web, esto sería de gran utilidad. Además, hace falta una mejor integración con los entornos de desarrollo para Java.

Una diferencia fundamental entre las aplicaciones desarrolladas utilizando GWT y las aplicaciones tradicionales HTML es que, mientras las aplicaciones GWT se ejecutan, no necesitan traer nuevas páginas HTML desde el servidor. No obstante, como todas las aplicaciones cliente-servidor, las aplicaciones GWT usualmente necesitan traer datos desde el servidor mientras se ejecutan. El mecanismo de interacción con el servidor a través de la red se denomina llamada a procedimiento remoto *(RPC, Remote Procedure Call)*. El mecanismo de RPC que provee GWT hace fácil la tarea de pasar objetos Java desde y hacia el servidor vía HTTP.

4.2.2. Servidor

Al utilizar el GWT, el servidor se debe implementar en lenguaje Java. En el estudio del estado del arte se vio que este es el lenguaje más utilizado para la implementación de las plataformas, seguramente debido a su portabilidad en los diferentes sistemas operativos. Una ventaja de utilizar Java es que los componentes pueden empaquetarse en archivos JAR y ser distribuidos entre los usuarios.

El UIMA SDK incluye una implementación en Java del marco de

trabajo. La versión actual del SDK está enfocada principalmente al desarrollo en Java pero también incluye facilidades para desarrollar componentes en C++.

Persistencia

La plataforma debe mantener en persistencia un conjunto de datos. Hay varias opciones para implementar la persistencia: se pueden utilizar archivos de texto o en otros formatos como XML, bases de datos, etc. En nuestro caso, se utilizó persistencia en archivos XML. La razón por la cual se hizo esto es que UIMA utiliza este tipo de persistencia para guardar la información de los componentes y de esta forma se podían utilizar algunas funciones de UIMA para cargar los datos a partir de archivos. Además de los datos de los componentes, se guardan otros datos como la información sobre los recursos de la plataforma, la lista de componentes integrados, y los tipos que se pueden utilizar en el sistema de tipos de un componente.

Módulos de Análisis

Cada componente o módulo de análisis integrado en la plataforma, consta de tres partes: Descriptores en archivos XML, Recursos, e Implementación en archivo JAR.

En los descriptores se guarda toda la información del componente: nombre, autor, descripción, nombre de la clase que lo implementa, parámetros de configuración, sistema de tipos, y recursos que utiliza. Por un lado, se guarda para cada componente un descriptor XML que sigue la misma sintaxis que define UIMA. Además, cada componente puede contar con un archivo XML adicional que guarda valores que pueden tomar los parámetros, esto se hace por fuera de UIMA y el propósito es contar con valores para poder darle a elegir al usuario cuando vaya a utilizar el componente.

El hecho de contar con los descriptores en el mismo formato que utiliza UIMA, facilita la tarea de «cargar» toda la información de un componente ya que se puede utilizar una funcionalidad de UIMA a la cual se le pasa el nombre del descriptor y carga en memoria toda la información del componente.

Puede pensarse que al utilizar los descriptores de UIMA se depende aún más de esta plataforma. Sin embargo, dado que es simplemente una forma de guardar la información en archivos XML, si se dejara de utilizar UIMA, estos archivos se pueden seguir manteniendo.

Los componentes pueden utilizar herramientas externas, por ejemplo, FreeLing [9]. Para esto, la plataforma guarda en un archivo XML la información sobre las herramientas externas existentes (rutas a sus archivos) para que los componentes puedan tener acceso a las mismas. Además, en el mismo archivo se guarda la información de los recursos globales que utilizan los componentes.

Existen además otros archivos XML que la plataforma utiliza para guardar información. En primer lugar, se tiene un archivo con la lista de componentes existentes. Por otro lado, se tiene una lista para definir los tipos que utiliza un componente, así como también sus atributos. Este último archivo nunca se modifica. Sin embargo, si en el futuro se quiere cambiar la lista de tipos permitidos, esto se hace fácilmente modificando el mismo.

La implementación de un componente se empaqueta en un archivo JAR con el mismo nombre del componente y contiene un archivo principal que implementa el componente utilizando UIMA y todos los archivos adicionales que utiliza.

4.2.3. Visualización de Resultados

Como se mencionó anteriormente, cada carácter en el texto, se considera una celda. El primer paso consiste en recorrer el conjunto de anotaciones existentes y determinar las propiedades de cada celda.

Luego de tener la información de cada celda, es decir, si tiene bordes y sus colores (de fondo y de borde), se procede a generar la visualización. Para esto, se comparan las celdas consecutivas y se «agrupan» las que son iguales en el sentido de tener los mismos colores y bordes.

Cada grupo de celdas iguales consecutivas se trata como una etiqueta o *label* utilizando el lenguaje del GWT. En lenguaje HTML, cada etiqueta se considera un *span*.

Para que las etiquetas tengan el color y bordes deseados, se utilizan las funciones que provee GWT para modificar el DOM *(Document Object Model)* del navegador [23]. Por ejemplo, mediante el siguiente código se crea un borde a la derecha en la etiqueta l:

```
DOM.setStyleAttribute(l.getElement(),"borderRight","solid "+color+" 2px")
```

Al implementar este algoritmo, surgió el problema del tiempo de procesamiento. Cuando se quieren ver los resultados de análisis de un texto muy grande con muchas anotaciones, se debe realizar mucho procesamiento en JavaScript y esto puede ocasionar que el navegador se bloquee y no muestre nada. Para solucionar este problema, se optó por dividir los resultados de análisis de un texto en varias páginas.

El texto se analiza completamente en el servidor, pero el resultado se visualiza en el cliente por partes, según la demanda del usuario. Esto soluciona el problema para los textos largos, a pesar de que puede ser molesto tener que esperar unos segundos para que se cargue la visualización.

Capítulo 5

Conclusiones y Trabajo Futuro

El proyecto consiste en el desarrollo de un ambiente web para procesamiento de lenguaje natural. Este ambiente se enfoca principalmente a la utilidad, de manera de permitir a usuarios tanto del área informática como de otras áreas, el acceso a un conjunto de funcionalidades para el análisis de texto. Por otra parte, el ambiente es la interfaz de usuario de una plataforma donde se aspira a la integración de módulos de análisis que encapsulen herramientas existentes, permitiendo el encadenamiento de módulos para realizar análisis complejos. Para poder encadenar módulos de análisis se debe contar con un estándar común para la representación de textos, una estructura que será compartida por los módulos de la cadena.

Se estudiaron las plataformas de PLN existentes respecto a sus arquitecturas de organización, estándares para representación de texto, visualización de resultados, etc. Al finalizar el estudio se obtuvieron tres conclusiones importantes. En primer lugar, existen varias plataformas que no tienen en cuenta los distintos tipos de usuario que necesitan hacer uso de estos sistemas. En particular, la plataforma UIMA no tiene una interfaz de usuario propia, ésta se ejecuta como un *plugin* del entorno de desarrollo Eclipse. Entonces, si un usuario quiere analizar un texto utilizando UIMA, necesita saber como funciona Eclipse, un entorno de desarrollo utilizado por muchos programadores, pero probablemente no conocido por muchos lingüistas. La segunda conclusión obtenida tiene que ver con el estándar para representación de textos que utilizan las plataformas. Se vio que la forma de representar el texto y los resultados es bastante similar entre los sistemas. El problema aquí es que no es igual, y en muchos casos, tampoco es compatible, lo que dificulta el encadenamiento de módulos que utilizan distinta representación. Por último,

se concluyó que existen carencias en cuanto a la visualización de resultados de análisis. Si bien existen plataformas que proveen métodos de visualización avanzados, éstos no son genéricos. Se vio que existe la necesidad de contar con una forma de visualización que permita cubrir los distintos tipos de análisis y que además tome en cuenta los problemas que surgen al mostrar anotaciones, como el solapamiento.

Dado que ya existen varias plataformas de PLN, se concluyó que no era razonable desarrollar por completo una nueva. Entonces, se desarrolló un sistema denominado Lavinia, basado en UIMA, una de las plataformas ya existentes. Lavinia utiliza la definición de módulo de análisis que define UIMA, y además la forma de representación del análisis de texto. Este estándar de representación será utilizado en el futuro por los usuarios de Lavinia, al desarrollar nuevos módulos de análisis.

Lavinia sigue una arquitectura cliente-servidor donde la interfaz de usuario es web y desarrollada utilizando el marco de trabajo *Google Web Toolkit* (GWT). La interfaz de usuario parece ser intuitiva y fácil de utilizar, pensada para que la utilicen usuarios que no tienen por qué ser del área de computación, por ejemplo lingüistas. Dado que la interfaz fue desarrollada utilizando el GWT, se basa en la nueva generación de páginas web denominada web 2.0. Se trata de páginas web dinámicas, orientadas al usuario, donde las actualizaciones se realizan en partes sin necesidad de recargar toda la página.

Se integraron módulos de análisis a Lavinia que cumplen algunas de las principales tareas de procesamiento de lenguaje natural. Por otra parte, se integró un módulo de análisis que no estaba dentro del alcance del proyecto. Este módulo estaba en el marco del curso Introducción al Procesamiento de Lenguaje Natural [15]. El módulo realiza un análisis semántico, desambiguando un conjunto de palabras y asignándoles a cada una un sentido obtenido de la base de datos léxica *WordNet*. Esta experiencia se consideró muy positiva para el ambiente ya que pudo comprobarse la facilidad de uso, tanto al momento de integrar el módulo, que en este caso era algo complejo ya que usaba recursos externos (*WordNet* y *Google*), como al momento de utilizarlo para analizar texto.

Finalizado el proyecto, se destacan dos aportes importantes. En primer lugar, se desarrolló un ambiente web para PLN. Esto resulta ser una novedad, ya que en la actualidad, los sistemas de PLN son aplicaciones de escritorio. El hecho de contar con un ambiente web, permite que usuarios desde cualquier parte del mundo, puedan integrar sus propios módulos de análisis y utilizarlos en cadena con

otros para realizar análisis complejos. Además, permite a los usuarios utilizar herramientas sin necesidad de realizar su instalación ni configuración, simplemente accediendo a la página web. El ambiente está pensado para que lo utilicen personas que no son necesariamente del área informática.

El siguiente aporte importante tiene que ver con la visualización de resultados implementada. Para su desarrollo se tuvieron en cuenta ciertos problemas como el solapamiento de anotaciones y se utilizaron prioridades para los tipos de anotaciones. Se obtuvo como resultado una buena solución que resultó ser innovadora y parece ser intuitiva para los usuarios.

En conclusión, se cumplieron los objetivos planteados al inicio del proyecto, y se obtuvo un resultado más que aceptable. Se logró desarrollar un ambiente web de procesamiento de lenguaje natural que puede ser de gran utilidad para personas que desarrollan nuevos módulos de análisis, y para personas que requieren analizar lenguaje natural, de manera sencilla. Además, los módulos de análisis pueden encapsular herramientas de procesamiento ya existentes que utilizan otros estándares de representación.

Como se mencionó anteriormente, el hecho de permitir agregar y modificar módulos desde la interfaz de usuario, puede traer problemas de seguridad. Este problema fue parcialmente solucionado al liberar una versión reducida de Lavinia, con un número acotado de funcionalidades, apta para ser publicada en internet sin problemas de seguridad. Como trabajo a futuro sería interesante contar con una solución a este problema. En particular, contando con distintos niveles de acceso según el usuario, este problema quedaría resuelto, restringiendo las funcionalidades mencionadas sólo a los usuarios «administradores».

En el ambiente web, cuando un usuario analiza un texto, tiene la opción de exportar los resultados a un archivo en formato XML. Relacionado a esto, otra funcionalidad que sería útil agregar es la de importar resultados de análisis. Puede pasar que un análisis lleve mucho tiempo y el usuario quiera hacerlo en etapas. Si el análisis consiste en el pasaje del texto por los módulos $M_1, M_2, \ldots, M_n$, sería interesante realizar el análisis utilizando $M_1, \ldots, M_j$, guardarlo, y luego levantarlo para continuar con $M_{j+1}, \ldots, M_n$. Para implementar esta funcionalidad, el usuario debe enviar al servidor los resultados y la información referente al sistema de tipos utilizado.

Actualmente la plataforma permite analizar texto ingresado desde la web o a partir de archivos que envía el usuario. Sería útil contar con otros mecanismos de importación de texto, por ejemplo, desde una dirección URL. Para implementar esto, se debe contar con un

mecanismo que permita obtener el texto de una página web, esto es, eliminando las marcas HTML, imágenes, etc. Contando con este mecanismo, el texto luego se puede analizar como se hace actualmente. Por otro lado, además de visualizar el resultado de análisis obtenido del texto de la página web, puede ser interesante visualizarlo con las marcas HTML y todo lo que la página que tenía. Esto implica implementar un mecanismo para quitar marcas HTML, analizar, y volver a colocar las marcas.

Glosario

AE Analysis Engines. Módulo de análisis en UIMA.

ANNIE A Nearly-New Information Extraction System. Conjunto de módulos provistos por la plataforma GATE, utilizados en aplicaciones de extracción de información.

ATLAS Architecture and Tools for Linguistic Analysis Systems. Una de las primeras arquitecturas desarrollada para sistemas de PLN.

Browser Explorador, navegador.

CAS Common Analysis Structure. Estructura utilizada por la arquitectura UIMA para que los módulos de análisis compartan la información de anotaciones, etc.

CPE Collection Processing Engine. Tipo de AE específico de UIMA encargado de procesar colecciones de documentos.

CSS Las hojas de estilo en cascada (Cascading Style Sheets, CSS) son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML).

DHTML El HTML Dinámico o DHTML designa el conjunto de técnicas que permiten crear sitios web interactivos utilizando una combinación de lenguage HTML estático, un lenguaje interpretado en el lado del cliente (como JavaScript) y el lenguaje de Hojas de estilo en cascada (CSS).

Framework En el desarrollo de software, un framework (marco de trabajo) es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado.

GATE General Architecture for Text Engineering. Marco de trabajo y entorno de desarrollo para PLN.

JAPE Java Annotations Pattern Engine. Lenguaje para expresar transducciones de estado finito mediante expresiones regulares.

JAR Formato de archivo utilizado para reunir todos los componentes que requiere una aplicación de Java.

Markup Cualquier cosa que se agregue al contenido de un documento que describe el texto. SGML y XML son lenguajes de markups.

Popup Elemento emergente, ventana flotante desplegada en pantalla, sobre la página visitada.

Refactorización Técnica de ingeniería de software para restructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

Renderizar Es la acción de asignar y calcular todas las propiedades de un objeto antes de mostrarlo en pantalla.

RGB Es el acrónimo inglés Red, Green, Blue (Rojo, verde, Azul). Es un modelo de color en el cual es posible representar un color mediante la mezcla de tres colores primarios: rojo, verde y azul.

SDK Kit de Desarrollo de Software

SGML Standard Generalized Markup Language. Lenguaje de marcas genérico, que al añadirle una semántica puede tener muy diferentes usos. Se usa en formatos de edición de documentos y en su aplicación más importante, el HTML.

Software Programas, procedimientos y reglas para la ejecución de tareas específicas en un sistema de cómputo.

String Conjunto de caracteres tratados como una unidad.

Tokenizar Es la acción de separar un string de caracteres en un conjunto de tokens.

Token Unidad mínima de un lenguaje.

UIMA Unstructured Information Management Architecture. Plataforma para PLN desarrollada por IBM

URL Significa Uniform Resource Locator, es decir, localizador uniforme de recurso.

XMI XML Metadata Interchange. Un modelo abierto para el intercambio el cual permite a desarrolladores que trabajan con objetos, intercambiar datos de programas de forma estandarizada.

XML eXtensible Markup Language. Lenguaje de marco ampliable o extensible desarrollado por el World Wide Web Consortium (W3C)

Referencias

[1] Next Generation Web Applications. Echo2. http://www.nextapp.com/platform/echo2/echo/. (Último acceso: mayo de 2007).

[2] Architecture and Tools for Linguistic Analysis Systems). Sitio oficial en internet. (Último acceso: junio de 2006).

[3] Denken Über. Ajax un nuevo acercamiento a aplicaciones web. http://www.uberbin.net/archivos/internet/ajax-un-nuevo-acercamiento-a-aplicaciones-web.php. (Último acceso: junio de 2007).

[4] Princeton University Cognitive Science Laboratory. Wordnet, a lexical database for the english language. http://wordnet.princeton.edu/. (Último acceso: julio de 2007).

[5] Sebastián Martínez Daniel Castelo, Jorge Isi. Webqa, respuesta automática a preguntas. Technical report, Instituto de Computación, Facultad de Ingeniería, Montevideo, Uruguay, 2007.

[6] Maestros del Web. ¿qué es la web 2.0? http://www.maestrosdelweb.com/editorial/web2/. (Último acceso: abril de 2007).

[7] Universitat Pompeu Fabra. Barcelona. Eduardo Sosa. Procesamiento del lenguaje natural: revisión del estado actual, bases teóricas y aplicaciones (parte i). http://www.elprofesionaldelainformacion.com/contenidos/1997/enero/procesamiento_del_lenguaje_natural_revision_del_estado_actual_bases_teoricas_y_aplicaciones_parte_i.html, 1997. (Último acceso: junio de 2007).

[8] Ellogon. Sitio oficial en internet. (Último acceso: junio de 2006).

[9] FreeLing. An Open Source Suite of Language Analyzers. http://garraf.epsevg.upc.es/freeling/. (Último acceso: enero de 2007).

[10] GATE. Sitio oficial en internet. http://gate.ac.uk/. (Último acceso: mayo de 2006).

[11] Google. Google web toolkit - build ajax apps in the java language. http://code.google.com/webtoolkit/. (Último acceso: mayo de 2007).

[12] Cutberto Uriel Paredes Hernández. Procesamiento computacional del lenguaje natural. http://www.monografias.com/trabajos5/proco/proco.shtml#apli, 2000. (Último acceso: junio de 2007).

[13] Laura Alonso i Alemany. Procesamiento del discurso. Technical report, PLN-FaMAF, Universidad Nacional de Córdoba, Argentina, 2006.

[14] IBM. Uima, unstructured information management architecture. http://www.research.ibm.com/UIMA/. (Último acceso: febrero de 2007).

[15] Facultad de Ingeniería Instituto de Computación. Introducción al procesamiento de lenguaje natural. http://www.fing.edu.uy/inco/cursos/intropln/. (Último acceso: julio de 2007).

[16] Universidad Europea de Madrid José María Gómez Hidalgo. Procesamiento del lenguaje natural. http://www.esp.uem.es/~jmgomez/pln/index.html. (Último acceso: julio de 2007).

[17] LinguaStream. An integrated experimentation environment for computational linguistics. http://www.linguastream.org/. (Último acceso: abril de 2007).

[18] Eclipse Project. Sitio oficial en internet. http://www.eclipse.org/. (Último acceso: junio de 2006).

[19] Grishman R. Tipster text architecture design. Technical report, New York University, 1998.

[20] Prototype Core Team. Prototype, javascript framework. http://www.prototypejs.org/. (Último acceso: junio de 2007).

[21] Raquel Martínez Unanue. Ingeniería lingüística aplicada al procesamiento de documentos. Technical report, Universidad Complutense de Madrid, España, 2004-2005.

[22] W3C. Cascading style sheets. http://www.w3.org/Style/CSS/. (Último acceso: abril de 2007).

[23] W3C. Document object model (dom). http://www.w3.org/DOM/. (Último acceso: abril de 2007).

[24] Wikipedia. Ajax. http://es.wikipedia.org/wiki/AJAX. (Último acceso: junio de 2007).

[25] Wikipedia. Dynamic web page. http://en.wikipedia.org/wiki/Dynamic_web_page. (Último acceso: junio de 2007).

[26] Wikipedia. Web 2.0. http://es.wikipedia.org/wiki/Web_2.0. (Último acceso: abril de 2007).