UTILIZACIÓN DE DATOS NO ESTRUCTURADOS COMO INTERFAZ DE OPERACIÓN EN APLICATIVOS DE CUALQUIER ÁMBITO, UTILIZANDO UIMA.

DANIEL JIMÉNEZ ORTIZ

UNIVERSIDAD DE CARTAGENA
FACULTAD DE INGENIERÍAS
PROGRAMA DE INGENIERÍA DE SISTEMAS
CARTAGENA – COLOMBIA
2011

UTILIZACIÓN DE DATOS NO ESTRUCTURADOS COMO INTERFAZ DE OPERACIÓN EN APLICATIVOS DE CUALQUIER ÁMBITO, UTILIZANDO UIMA.

DANIEL JIMÉNEZ ORTIZ
. N

Trabajo de grado para optar el título de Ingeniero de Sistemas

ASESOR
LUIS CARLOS TOVAR GARRIDO
INGENIERO DE SISTEMAS

UNIVERSIDAD DE CARTAGENA
FACULTAD DE INGENIERÍAS
PROGRAMA DE INGENIERÍA DE SISTEMAS
CARTAGENA – COLOMBIA
2011

PAGINA DE ACEPTACIÓN

Nota de aceptación

_____

_____

_____

_____

_____

_____

_____

Firma del Jurado

Cartagena, 21 de Junio de 2011

AGRADECIMIENTOS

Agradezco a mi profesor Julio Rodriguez Ribón quien me enseñó que la respuesta a los problemas está en buscar la solución, al profesor Martín Emilio Monroy Ríos quien demostró que con una clara concepción de los conceptos se puede lograr una implementación impecable y al Ingeniero Luis Carlos Tovar Garrido que me mostró, como con la ciencia e ingeniería se entrelazan para mejorar muchos aspectos de la condición humana.

CONTENIDO

ILUSTRACIONES

SIGLAS

IPO: Interacción Persona Ordenador.

UIM: Unstructured Information Management.

UIMA: Unstructured Information Management Arquitecture.

PLN: Procesamiento del Lenguaje Natural.

AE: Analysis Engine.

UJI: Uima Java Interface.

CAS: Common Analysis System.

OASIS: Organization for the Advancement of Structured Information.

GLOSARIO

ANÁLISIS MORFOLÓGICO: consiste en determinar la forma, clase o categoría gramatical de cada palabra de una oración.

APIS: interfaz de programación de aplicaciones (Applications Programming Interface): una serie de funciones que están disponibles para realizar programas para un cierto entorno.

APLICACIÓN CLIENTE: es el software que será operado mediante la arquitectura.

ARQUITECTURA DE SOFTWARE: según el documento de IEEE Std 1471-2000, "La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución".

EVENTOS DEL OPERATIONDESCRIPTOR: cada método del artefacto está relacionado con un evento. Un evento es una característica que podrá ser encontrada en la información no estructurada utilizada como entrada, y de ser así el evento se ejecutará.

EXPRESIÓN REGULAR: es una forma de representar a los lenguajes regulares (finitos o infinitos) y se construye utilizando caracteres del alfabeto sobre el cual se define el lenguaje.

EXTRACCIÓN DE INFORMACIÓN (EI): consiste en extraer, de un texto.

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.

INTERACCIÓN MULTIMODAL: es un proceso en el cual diversos dispositivos y personas son capaces de llevar a cabo una interacción (auditiva, visual, táctil y gestual) conjunta desde cualquier sitio, en cualquier momento, utilizando cualquier dispositivo y de forma accesible, incrementando así la interacción entre personas, y entre dispositivos y personas (W3C Multimodalidad, 2011).

PATRÓN: representación abreviada de la secuencia consenso de un motivo, por ejemplo, expresiones regulares.

POS-TAGGER: programa de etiquetado léxico cuya finalidad es la de indicar que función cumple cada una de las palabras dentro de la oración.

PROCESAMIENTO DE LENGUAJE NATURAL: se ocupa de la formulación e investigación de mecanismos eficaces computacionalmente para la comunicación entre personas o entre personas y máquinas por medio de lenguajes naturales.

RECONOCEDOR DE ENTIDADES (NAME ENTITY TAGGER): sistema capaz de detectar y clasificar entidades nombradas en fragmentos de texto.

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.

XML: eXtensible Markup Language, es el lenguaje de marco ampliable o extensible desarrollado por el World Wide Web Consortium (W3C, 2011).

1. RESUMEN

Frente a la necesidad de masificar el uso de aplicativos operados mediante métodos no convencionales, tales como la voz, el video o el lenguaje natural; y reconociendo que en la actualidad existen organizaciones y/o grupos de investigación, que han realizado notables adelantos en el área del procesamiento de información no estructurada y en arquitecturas, frameworks y herramientas que facilitan y estandarizan este proceso; esta investigación presenta una forma de aprovechar todos estos adelantos y ofrecerlos a constructores de software, para que puedan anexarlos a sus proyectos y de esta manera brindar la posibilidad a sus usuarios finales de poder operar sus aplicativos mediante datos no estructurados.

En este proyecto se creó una herramienta que funciona como puente entre una arquitectura para el manejo de la información no estructurada denominada UIMA y una aplicación de cualquier ámbito o tamaño. Concretamente se construyó un middleware implementado en Java, que se anexa en forma de librería, y que ejecuta los diferentes métodos del aplicativo del constructor de software, dependiendo del resultado del procesamiento de la información realizada por la arquitectura.

Al final del proceso se obtuvo una herramienta diseñada para constructores de software denominada UJI (Uima Java Interface) y un prototipo que sustenta su

funcionalidad. Como aporte importante se destaca la facilidad de uso, limitando la configuración de la interfaz a un archivo xml estándar.

Como UJI utiliza UIMA, y esta arquitectura es considerada un estándar para el procesamiento de información no estructurada, todos los aportes que se hagan al área estarán beneficiando a UJI.

La importancia de este trabajo yace en el hecho que UJI facilita de gran manera, la tarea de implementar software operados mediante métodos no convencionales, y esto se refleja en aspectos prácticos tales como que los constructores de software podrían crear más y mejores aplicativos dirigidos a personas con discapacidades físicas, nuevos paradigmas de usabilidad, innovadoras interfaces de usuario, entre otros beneficios que se mencionarán a lo largo de este documento.

Palabras Claves: UIMA, Interacción persona-ordenador, Procesamiento de Lenguaje Natural, Datos no estructurados.

ABSTRACT

Faced with the need to expand the use of applications operated by unconventional methods, such as voice, video, or natural language, and recalling that at present there are organizations and / or research groups who have made significant advances in area of unstructured information processing and architectures, frameworks and tools that facilitate and standardize this process, this research presents a way to get all these developments and offered to software developers so they can append to their projects and thus provide the possibility to end-users to run their applications with unstructured data.

In this project has be create a tool that works as a bridge between an architecture to manage unstructured information called UIMA and an application any scope or size. Specifically, we built a middleware implemented in Java, which is attached as a library, and runs the different methods of application builder software, depending on the outcome of information processing by the architecture.

At the end of the process obtained a tool designed for developers of software called UJI (UIMA Java Interface) and a prototype that supports its functionality. As an important contribution highlights the ease of use, limiting the configuration of the interface to a standard xml file.

As UJI using UIMA, and this is considered a standard architecture for processing unstructured information, all contributions made to the area are benefiting UJI.

The importance of this work lies in the fact that UJI greatly facilitates the task of implementing software operated by unconventional methods, and this is reflected in practical aspects such as which software developers could create more and better applications aimed at people with physical disabilities, new paradigms of usability, innovative user interfaces, among other benefits which are considered throughout this document.

Keywords: UIMA, Human-Computer Interaction, Natural Language Processing, unstructured data.

2. INTRODUCCIÓN

Cuando los seres humanos y los ordenadores interactúan lo hacen a través de un medio o interfaz. En el caso de la Interacción Persona Ordenador, la interfaz es el punto en el que seres humanos y ordenadores se ponen en contacto, transmitiéndose mutuamente tanto información, órdenes y datos como sensaciones, intuiciones y nuevas formas de ver las cosas (Lorés, 2002). La interacción Persona Ordenador (IPO), es una disciplina relacionada con el diseño, evaluación e implementación de sistemas informáticos interactivos para el uso de seres humanos, y con el estudio de los fenómenos más importantes con los que está relacionado.

Una de las áreas de estudio de la IPO es la Interacción lingüística oral persona-ordenador (Diaper & Neville, 2004), que representa una de las mejores formas de establecer una comunicación entre humanos y máquinas, debido a que el lenguaje oral es el medio más utilizado por las personas para transmitir información, y al darle esta facultad a los ordenadores se estaría llevando la interacción con estos hasta un nivel de fluidez casi natural (Gómez, 2000).

En la Ilustración 1 se muestra de manera general como funciona el proceso de la interacción lingüística oral persona-ordenador, en donde este inicia con la entrada de un enunciado oral generado por el usuario, que luego es analizado

por un componente de identificación de lengua y después a uno de reconocimiento del habla, que convierte de forma automática la voz a texto, de forma inmediata este texto sirve como entrada para el componente de comprensión del lenguaje, basado en una de las ramas de la inteligencia artificial el Procesamiento del Lenguaje Natural (PLN) analiza de forma morfológica, sintáctica, semántica y pragmática, para luego generar un resultado estructurado del significado de la entrada y suministrarlo a la aplicación informática que lo solicite.

Luego que la aplicación procesa la información, su resultado es enviado a un componente que la transforma en un texto expresado en lenguaje natural que pueda ser comprendido por el usuario, iniciando el flujo en el sentido inverso, luego este texto es sintetizado al formato de habla en la lengua correspondiente a la entrada del texto y finalmente es suministrado al usuario que generó la entrada.

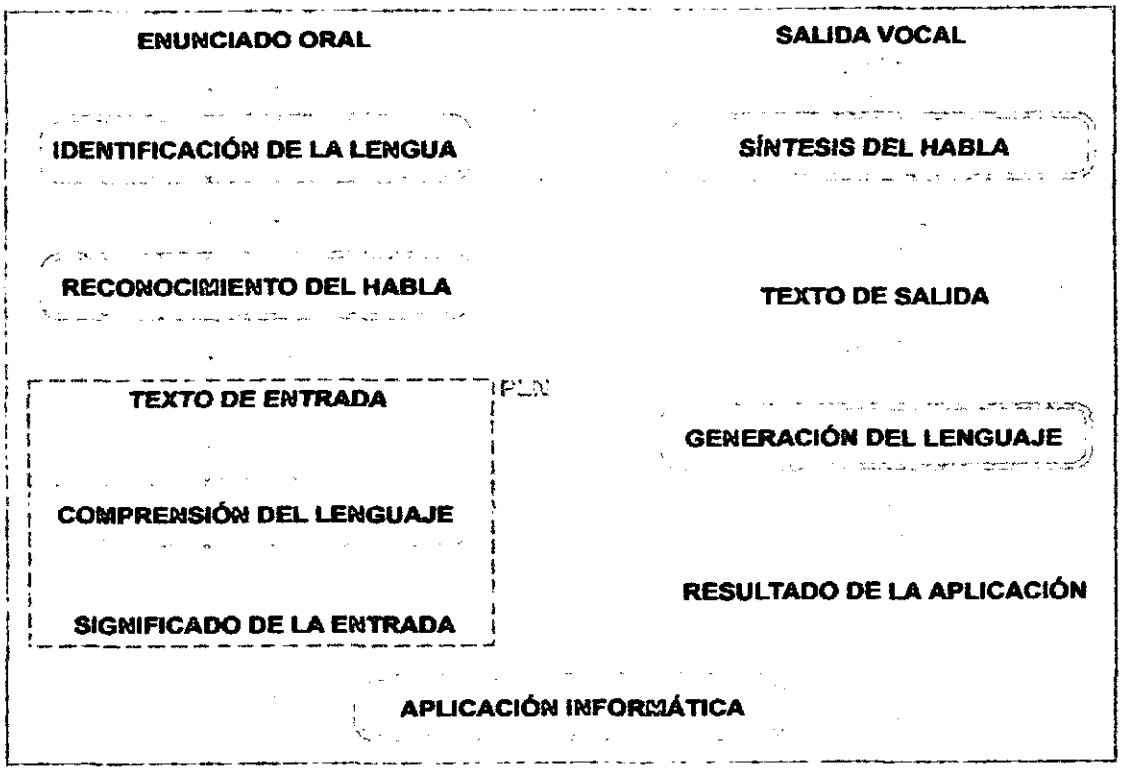


Ilustración 1: Interacción lingüística oral persona-ordenador. Fuente: Introducción al procesamiento del lenguaje natural. Moreno B, Universidad de Alicante, 1999

El lenguaje oral como forma de interacción entre personas y ordenadores es solo una parte del procesamiento de la información multi-modal como medio para comunicarnos con las máquinas, ya que además de éste existe todo tipo de información no estructurada (Malone, 2007), es decir documentos cuya intención o significado no se encuentran explícitos y que es necesario analizarlos minuciosamente para detectar sus principales componentes, las relaciones entre estos y su relación con el aplicativo informático que los requiere, tales como imágenes, videos o textos.

Una aplicación gestora de información no estructurada (UIM por sus siglas en ingles Unstructured Information Management) (Apache UIMA Development Community, 2009) es un software que analiza grandes volúmenes de información no estructurada (textos, videos, imágenes, entre otros.), para descubrir, organizar y entregar conocimiento relevante al cliente o al usuario final (Ferruci & Lally, 2004). De esta manera para crear un aplicativo multi-modal para la IPO es necesario implementar una UIM, de tal forma que independientemente del formato de entrada de información, el aplicativo informático de procesamiento siempre cumpla su función.

En la actualidad existe una serie de herramientas y/o UIM´s que le permiten a los constructores de software, organizar, analizar y procesar información no estructurada explícitamente, tales como videos, imágenes, sonidos o textos, de tal forma que se puede extraer sus principales componentes y la relación entre estos, lo que conlleva a obtener información con una estructura formal. Un ejemplo de su utilidad es hallar el significado de un documento de texto escrito en lenguaje natural o establecer una relación entre un conjunto de imágenes aleatorias.

Estas herramientas brindan a otras aplicaciones la posibilidad de acceder a sus análisis de información no estructurada, lo que significa que otros software pueden beneficiarse de su poder de procesamiento y utilizar los datos procesados (información estructurada) para su propio funcionamiento, facilitando la construcción de aplicaciones gestoras de información

estructurada, tales como motores de búsqueda semánticos o traductores automáticos.

Sin embargo, pese a que existe un gran número de investigaciones realizadas en el área del análisis de información no estructurada (lingüística computacional, minería de imágenes, minería de sonidos, entre otros.) (Gómez, 2000), además de la amplia utilización de herramientas que permiten integrar estos avances con otros aplicativos, No es significativo el aprovechamiento de dichos adelantos en otras áreas de la computación, tales como la usabilidad y/o la Interacción persona-ordenador, ya que es muy escaso la literatura que documente la utilización de forma sencilla en dichas áreas.

Una herramienta para crear fácilmente aplicativos operados mediante datos no estructurados es el objetivo central de esta investigación, fundamentada en el hecho de que una herramienta de este tipo podría impulsar la creación y mejoramiento de aplicativos con nuevas e innovadoras interfaces de usuario.

En pocas palabras, mediante la creación de una herramienta con las características mencionadas en el párrafo anterior, se les facilita a los constructores de software, la manera de hacer aplicaciones manejadas mediante la texto, audio, imágenes, videos o cualquier tipo de información no estructurada. A continuación se mencionan algunos posibles ejemplos de aplicaciones:

- Aplicativos operados mediante la voz, para personas sin miembros superiores o con problemas de visión.
- Software manejados mediante gestos, para la agilización de tareas.
- Aplicaciones autómatas controladas por video, para el control de procesos industriales repetitivos o peligrosos.
- Interfaces de usuario para terminales bancarias, guiadas por el reconocimiento de rostros introducido por un flujo de video.
- Interfaces de usuario multimodales, para mejorar la experiencia en aplicaciones turísticas.
- Interfaces adaptatívas a los estados anímicos de los usuarios.
- Aplicativos domóticos.

2.1. ANTECEDENTES

En el sector de procesamiento de información no estructurada, se han hecho aportes muy significativos, ya que existen frameworks y componentes reutilizables que dan soporte a la necesidad de interpretar la información proveniente de fuentes tan comunes, como el texto, audio o video.

Entre los frameworks más importantes se encuentran:

1. GATE (Cunningham et al., 2002): Un framework y un entorno de desarrollo gráfico que permite a los usuarios desarrollar y desplegar componentes de lingüística computacional y recursos de una manera robusta .

2. Ellogon (Petasis et al., 2002): Un multi-idioma, multi-plataforma, ambiente de ingeniería de texto de uso general. Ellogon fue diseñado con objeto de ayudar tanto a los investigadores en el procesamiento de lenguaje natural, así como las empresas que producen sistemas de ingeniería lingüística para el usuario final. Ellogon proporciona una potente infraestructura TIPSTER basado en la gestión, almacenamiento e intercambio de datos de texto, integración y gestión de componentes de procesamiento de textos, así como visualizar datos de texto y la información lingüística asociada.

3. LinguaStream (Bilhaut y Widlöcher, 2007): Es un framework que facilita el procesamiento complejo de flujos de datos. Este permite el montaje componentes de análisis de diversos tipos y niveles: etiquetado gramatical, sintaxis, semántica, discurso o statistical .

En el 2006 IBM desarrolla UIMA (Unstructured Information Management Architecture), una plataforma que permite la creación e integración de

aplicaciones que procesen información no estructurada como textos en lenguaje natural, audio o vídeo (Cruz et al., 2009), posibilitando el proceso de diseñar aplicativos que automáticamente analicen y extraigan conocimiento de bancos de información.

Además existen un considerable número de componentes reutilizables, disponible para UIMA en la red, como motores de análisis, tokenizadores, typesystems, CAS, Collection Readers, y todo tipo de módulos que sustentan los procesos de PLN (Procesamiento del Lenguaje Natural), generadores automáticos de resúmenes, herramienta para la traducción entre otros (Jena University, 2010; Hahn et al., 2008; Techera, 2007).

El área de la interacción persona-ordenador nos encontramos con la investigación titulada "Processing dialogue-based data in the UIMA framework", de los doctores Milan Gnjatović, Manuela Kunze y Dietmar Rösner (Gnjatović et al., 2008), quienes plantean que para establecer una verdadera interacción entre humanos y computadores es necesario utilizar como medio de comunicación el dialogo común, y para ello se requiere procesar diferentes formas de datos, como video, audio, textos, etc. En dicha investigación se usó la plataforma UIMA, para aprender del resultado estadístico de las ocurrencias de patrones lingüísticos y su incidencia en el comportamiento de la interacción persona-ordenador.

Actualmente no existen frameworks o interfaces gratuitas, que faciliten a los

constructores de software implementar arquitecturas como GATE, Ellogon, linguastream o UIMA para procesar datos no estructurados y utilizar el resultado del procesamiento en la interacción persona-ordenador, de sus aplicativos.

2.2. PLANTEAMIENTO DEL PROBLEMA.

Estudios realizados (Gantz, 2007), revelan que en el año 2006 el volumen de transmisión de información mediante la Web estaba alrededor de los 161 billones de gigabytes, lo que equivale a unas tres millones de veces la información almacenada en todos los libros antes escritos, además esta tiene una tasa de crecimiento del 57% anual, lo que hace pensar que para el 2010 se estime un aproximado de 988 exabytes de información. Según Gantz casi un 95% de la información trasmitida mediante Internet está constituida por imágenes, sonidos, videos, música o documentos de textos simples.

Analizar estos bastos volúmenes de información resulta una tarea primordial para una verdadera sociedad del conocimiento; sin embargo debido a la magnitud de los flujos de información se hace necesario recurrir a la ayuda de aplicativos informáticos que los procesen y que generen bancos de conocimiento automáticamente.

Las versiones electrónicas de imágenes, sonidos, textos y sus variaciones como videos o música, pueden ser considerados por un ordenador como información no estructurada, debido a que no poseen una estructura explicita, en donde se identifiquen claramente sus principales componentes y sus relaciones, como es el caso contrario de un archivo XML o una ontología.

UIMA procesa documentos no estructurados mediante un motor de análisis o analysis engine (AE) y posteriormente genera una serie de deducciones, descritos en CAS (Common Analysis System), que son la representación estructurada del resultado del análisis, ofrecidas mediante interfaces a las diferentes aplicaciones que lo requieran. Para acceder y manipular los datos del CAS es necesario conocer los tipos y características definidos con anterioridad en el type system[1] (Apache UIMA Development Community, 2009), lo que conlleva a que el uso de la plataforma por parte de constructores de software inexpertos en el tema del procesamiento de información no estructurada, implique un considerado nivel de dificultad, ya que éstos además de sus aplicativos tendrían la necesidad de implementar componentes para el funcionamiento de UIMA, tales como descriptores del type system, de CAS y de AEs entre otros. Además seria necesario construir interfaces que adapten la plataforma con sus aplicativos.

1Define los diferentes tipos de objetos que podrán ser descubiertos por el AE, en el análisis del documento.

Con UIMA construir fácilmente aplicaciones de cualquier tamaño y ámbito que tengan la posibilidad de manejar como entrada operativa información no estructurada, resulta un inmenso adelanto en el área de la usabilidad, ya que el usuario puede operar la aplicación mediante el uso del lenguaje natural o de cualquier otro tipo de información no estructurada, evitando forzarlo al aprendizaje de una determinada interfaz de operación (Diaper & Neville A, 2004). Pero esta intención se ve cercenada por el alto nivel de complejidad al momento integrar la plataforma a la aplicación, debido a que es necesario invertir gran esfuerzo y tiempo en la configuración, implementación y adaptación de la herramienta, ya que fue diseñada para expertos en el área del procesamiento de información no estructura; además el manejo del resultado del análisis conlleva un problema debido a que es necesario encontrar una forma para que los datos estructurados desencadenen una acción en otras aplicaciones.

En la actualidad las aplicaciones solo tienen la posibilidad de interactuar con el resultado obtenido de la interpretación de información no estructurada de forma analítica, como buscadores semánticos, traductores automáticos o generadores de resúmenes, entre otros (Burgos Domingéz, 2002), reduciendo el uso de estas tecnologías, y dificultando su implementación como una interfaz de manejo para aplicaciones que no están relacionadas con este sector.

UIMA representa una opción para que constructores de software, en sus diferentes niveles de capacitación puedan diseñar e implementar aplicativos

capaces de ser operados mediante información no estructurada (como por ejemplo aplicaciones operadas mediante la voz, video o textos en lenguaje natural), siempre y cuando se pueda encontrar una solución a la problemática del manejo de los resultados del análisis y a la complejidad de la configuración de la plataforma.

2.3. JUSTIFICACIÓN

Al facilitar la forma en que se construyen aplicativos, operados mediante información no estructurada, se está fomentando un ambiente para el desarrollo en varias áreas de la computación, debido a que:

1. Se está resolviendo en gran medida la necesidad implementar masivamente, diferentes formas de interactuar con aplicaciones informáticas(Lorés, 2002), en software de cualquier tamaño o ámbito, mediante la arquitectura UIMA.

2. Se está haciendo un gran adelanto en el área de la usabilidad, debido a que el usuario final puede operar aplicaciones mediante el uso del lenguaje natural o de cualquier otro tipo de información no estructurada, evitando forzarlo al aprendizaje de una determinada interfaz de operación (Diaper & Neville, 2004).

3. Se está facilitando a los desarrolladores de software una nueva forma de utilizar la plataforma UIMA.

Por otra parte, alcanzar estas metas es de relativa sencillez, debido a lo siguiente:

1. Desde el punto de vista económico es factible, ya que no es necesario adquirir ninguna licencia de software comercial, debido a que la plataforma UIMA es de licencia Apache (Apache Licence, 2011), se utilizará el lenguaje de programación JAVA (Goslin et al., 2000), de libre uso, el entorno de programación gratuito Eclipse (Eclipse, 2011), herramientas gratis de modelado como StarUML (StarUML, 2011), sistema operativo Linux de licencia GPL (GPL Licence, 2011) y el procesador de textos OpenOffice.

2. Desde el punto de vista tecnológico es viable, debido a que en la actualidad existe un repertorio de herramientas informáticas al alcance, que sustentarían el proceso de investigación e implementación de una solución, que en este caso sería de tipo conceptual y/o informático, como es el caso de lenguajes y entornos de programación, herramientas de modelado, framework, API´s, patrones de diseño y arquitectónicos, estructuras de datos entre otros.

3. Desde el punto de vista científico su viabilidad es muy alta, ya que está

contribuyendo a crear un puente de contribución entre dos sectores de las ciencias computacionales muy importantes, como la Interacción persona-ordenador y algunas ramas de la inteligencia artificial como la lingüística computacional y el procesamiento de imágenes, textos o sonidos.

En el ámbito local, una herramienta que contribuye a que los aplicativos puedan ser operados de innovadoras formas, es de gran importancia, debido a que junto a nuevas tecnologías como la realidad aumentada y la minería de textos, imágenes sonidos y videos, se podrían construir aplicaciones orientadas a la visualización y ubicación de atractivos turísticos, operadas mediante la voz, imágenes o simplemente un video tomado desde un celular; mejorando en gran medida la experiencia de turistas e innovando la forma en que las personas perciben los atractivos de la ciudad.

En el ámbito profesional, este proyecto aporta un valor agregado al proceso de elaboración de software, ya que posibilita la opción de anexar nuevas características de usabilidad a los aplicativos y de esta forma llevar a un nicho de mercado más amplio y generar mayor satisfacción en los usuarios finales, lo que puede conllevar a un mayor auge de las herramientas informáticas y consecuentemente un impulso a la profesión.

Por último, el aporte de una interfaz que pone al servicio de los desarrolladores, la oportunidad de utilizar mecanismos de interpretación de

datos no estructurados como UIMA, al momento de crear aplicativos que interactúan con el usuario de nuevas formas (movimientos, sonidos o gestos), mejora las condiciones de vida de las miles de personas del país, que poseen discapacidades que afectan la normal utilización de ordenadores. Ya no sería necesario la utilización de los miembros superiores, para operar los aplicativos, si no que se podrían manejar mediante comandos por voz, movimiento del ojo o la cabeza, algún tipo de gesto, o cualquier información que se pueda capturar de cualquier parte del cuerpo.

2.4. CONTEXTO DE LA INVESTIGACIÓN

La investigación tuvo lugar en la ciudad de Cartagena – Colombia, en las instalaciones de la Universidad de Cartagena, en el programa de Ingeniería de Sistemas. Inició el 31 de Agosto de 2010 y tuvo una duración de 6 meses (ver METODOLOGÍA, en la página 51).

3. MARCO TEÓRICO

3.1. UNSTRUCTURED INFORMATION MANAGEMENT ARCHITECTURE (UIMA)

UIMA (Unstructured Information Management Architecture) (Overview and Setup, 2009) es una plataforma de software extensible y escalable que permite ordenar y procesar datos no estructurados como texto, sonido, imagen, etc. El sistema fue inicialmente desarrollado por Alpha Works IBM aunque ahora es Apache Software Foundation (ASF, 2011) quien se encarga del mantenimiento, de las licencias y de las nuevas versiones. UIMA es un software abierto programado en lenguaje Java que permite al desarrollador trabajar en este mismo lenguaje o en C++. La arquitectura UIMA se basa en el desarrollo de módulos independientes de proceso de datos. Estos se pueden enlazar de forma que un módulo recibe información del módulo anterior y transmite nueva información al módulo siguiente.

Gracias a las herramientas de que dispone la plataforma UIMA se pueden utilizar arquitecturas distribuidas. Esto significa que cada uno de los módulos de proceso puede estar localizado en una máquina distinta y ser utilizado de forma remota. La plataforma UIMA cuenta con un conjunto de herramientas (UIMA Tools Guide and Reference) para facilitar el desarrollo de módulos,

componentes y arquitecturas completas. El software cuenta con distintos plugins para el entorno de programación libre Eclipse (Eclipse, 2011). Gracias a estos plugins se pueden editar ficheros fundamentales en UIMA de forma fácil y cómoda (Poch, 2007).

En el artículo donde se presentó UIMA (Ferrucci, 2004) se argumenta que IBM dispone de un grupo de alrededor de 200 personas a lo largo de todo el mundo, trabajando en los más diversos temas relacionados con el procesamiento del lenguaje natural, además afirma que al perfeccionar las habilidades de la organización, aumentando la velocidad y eficiencia de transmisión de información para descubrir otros resultados y rápidamente combinarlos con diferentes tecnologías, se estaría acelerando los avances científicos. Esto es lo que se considera el fundamento central de UIMA, una arquitectura estándar y un robusto software que de soporte a la ágil transferencia de resultados de investigaciones sobre el procesamiento del lenguaje natural, dentro de IBM.

La arquitectura para la gestión de información no estructurada UIMA, como su nombre lo indica es una arquitectura y un framework que ayuda a crear un puente que soporta la creación, descubrimiento, composición y el despliegue de un ancho rango de capacidades de análisis y conexión de los servicios de estructuración de la información.

3.1.1. ANALYSIS ENGINES (AEs).

UIMA es una arquitectura basada en bloques de trabajo llamados Motores de Análisis (AE), diseñados para analizar los diferentes documentos e inferir una serie de datos descriptivos relacionados con parte o la totalidad del documento. Esta información descriptiva producida por los AEs, generalmente es nombrada como "el resultado del análisis", que se representa en meta-datos acerca del contenido del documento original.(Overview and Setup, 2009)

Un motor de análisis puede representar una función granular para determinada tarea en el procesamientos de los datos, a lo que se le denomina *simple* o *primitive UIMA Analysis Engine (Overview and Setup 2009),* adicionalmente esta función puede hacer parte de un sistema más grande, y que este AE solo es un engranaje en el flujo de procesamiento, esperando la salida de un motor de análisis para que luego de su procesamiento se lo pase a otro. Este caso más complejo es llamado *Aggregate Analysis Engines* (motores de análisis agregados).

Un ejemplo práctico de motores de análisis agregados, es el caso de la construcción de un aplicativo para el procesamiento del lenguaje natural usando la plataforma UIMA, en este caso tendríamos que construir un pipeline o una estructura que contemplara el procesamiento de las principales tareas de

un PLN, que son:

1. Análisis morfológico.
2. Análisis sintáctico.
3. Análisis semántico.
4. Análisis pragmático.

En donde cada tarea estará incluida en un motor de análisis, tal y como se observa en la ilustración 2, de esta forma se encapsula su complejidad, se facilita su mantenimiento y se garantiza su reutilización entre otras ventajas. Lo siguiente que debemos hacer es establecer un flujo de procesamiento, que estructurará en qué forma y secuencia se realizará el procesamiento, en el caso de ejemplo es el siguiente:

1. El documento es enviado al motor de análisis morfológico, en donde se detecta la relación que se establece entre las unidades mínimas que forman una palabra y se graba en los meta-datos del documento.

2. El documento ahora es enviado al motor de análisis sintáctico, aquí inicialmente son leídos los meta-datos del anterior AE y con esta información se etiqueta cada uno de los componentes sintácticos que aparecen en la oración y se analiza cómo las palabras se combinan para formar construcciones gramaticalmente correctas. El resultado de este proceso genera la estructura correspondiente a las categorías sintácticas

formadas por cada una de las unidades léxicas que aparecen en la oración.

3. Ahora el flujo de información es dirigida hacia el motor de análisis semántico, en donde con las nuevas etiquetas anexadas a los meta-datos es posible intentar dar un significado al documento, de esta forma el motor por cada construcción gramatical u oración se graba una lista de posible significados.

4. En la última parte del proceso, el motor de análisis pragmático recoge toda la información descriptiva indexada a los meta-datos del documento y analiza la relación entre estos, para que de alguna manera encontrar un contexto para detectar cual de los significados es el más acertado para el documento, y nuevamente esta información es grabada y enviada para que una interfaz haga lo adecuado con esta información.

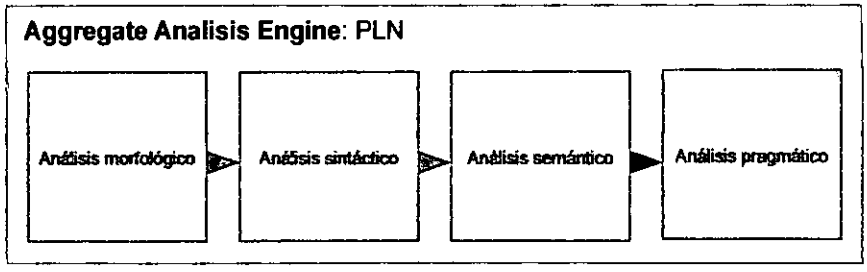


Ilustración 2: Ejemplo de un motor de análisis agregado

Continuando con el tema, los meta-datos o el resultado del análisis incluyen diferentes sentencias acerca del contenido de un documento. Tal y como se muestra en el siguiente ejemplo, en donde se explica el procesamiento de dos

documentos de diferente formato (uno de texto y otro de imagen), de manera transparente.

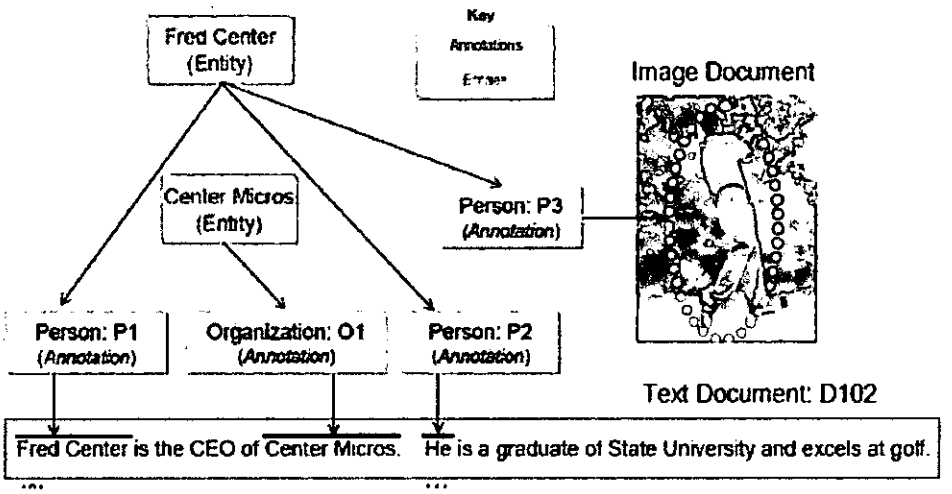


Ilustración 3: Ejemplo de representación de objetos (UIMATools Guide and Reference)

Una sentencia puede representar una región granular de la totalidad del documento, una secuencia de caracteres enmarcados en un principio y un fin denotados por el orden numérico de los caracteres donde empieza y termina dicha secuencia, y que es denominada *span.* Considerando el ejemplo de la Ilustración 3, una sentencia puede ser la siguiente: "El documento identificado con el nombre D102, contiene el span 'Fred Center', que inicia en el caractér 101 y termina en el 112, y este denota una persona".

Que en un lenguaje técnico sería:

(1) El span de la posición 101 a la 112 en el documento D102 denota una Persona

La última parte de la sentencia se refiere a una "Persona", este es un término especial predefinido que caracteriza un tipo de resultado, y que es detectado y etiquetado por los AEs. Estos términos son denominados *Types.*

Otro resultado del análisis puede relacionar dos sentencias, denotando que hacen referencia al mismo type. Como por ejemplo:

(2) La persona denotada por el span del 101 al 112 y
la persona denotada por el span 141 to 143 en el documento D102
se refieren a la misma Entidad.

Cuando se utiliza el término Entidad, se hace referencia a que es una instancia de un type, así mismo como lo es un objeto a una clase. En la ilustración 3, se muestra como dos documentos inclusive de diferentes formatos, pueden hacer referencia a una misma entidad.

NOTA: Los ejemplos de sentencias 1 y 2 no corresponden a una sintaxis específica de UIMA.

3.1.2. ANNOTATORS.

La plataforma UIMA provee un componente básico, que representa el núcleo de los algoritmos de análisis que se ejecutan dentro de los AE´s, y las instancias de este componente son denominados anotadores.

Los anotadores son los bloques de construcción que contienen la lógica del análisis, y realizan dentro de los AE´s el proceso de extracción y generación de meta-datos, con la información relacionada al documento o artefacto.

En la implementación, para construir un anotador solo basta con crear una clase java que implemente todos los métodos de la interfaz estándar AnalysisComponent, o simplemente heredar de JcasAnnotator_ImplBase, que contiene la implementación de los métodos necesarios para el análisis, a excepción del método abstracto process.

3.1.3. COMMON ANALYSIS STRUCTURE (CAS).

UIMA Common Analysis Structure es la estructura de datos primaria que los componentes de análisis UIMA utilizan para representar y compartir los

resultados de análisis.

1. El artefacto: Es el objeto que se analiza, como un documento de texto o flujo de audio o vídeo. Los CAS contienen uno o varios puntos de vista del artefacto. Cada vista se conoce como Subject of Analysis.

2. Una descripción del sistema de tipos que indica los tipos, subtipos, y sus características.

3. Análisis de los metadatos "standoff", anotaciones que describen el artefacto o de una región del artefacto.

4. Un índice del repositorio para facilitar el acceso eficiente e iteración sobre los resultados del análisis.

5. Interfaz principal UIMA, esta estructura es proporcionada por una clase llamada el *Common Analysis System*. Cuando la estructura de análisis común se utiliza a través de una interfaz diferente, la aplicación particular de la estructura se indica, por ejemplo, JCAS es un nativo de la representación del objeto de Java de los contenidos de la estructura de análisis común.

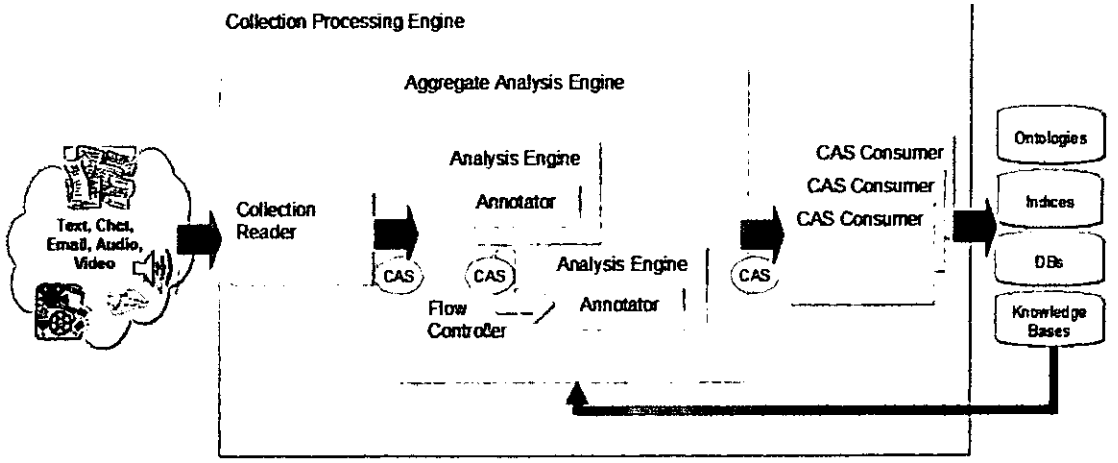


Ilustración 4: Arquitectura de Alto Nivel UIMA de componente,s de la fuente al sumidero (Overview and Setup 2009)

En la Ilustración 4 se puede visualizar la forma en que los diferentes componentes de UIMA interactúan entre sí mediante CAS, además al final del procesamiento, componentes externos pueden acceder a la información del resultado del análisis mediante los llamados CAS Consumer, que no son otra cosa que CAS con todas las anotaciones realizadas por los motores de análisis.

3.1.4. COMPONENT DESCRIPTORS.

UIMA define interfaces para un pequeño conjunto de componentes básicos, que los usuarios del framework pueden utilizar para proporcionar sus implementaciones . Motores de análisis y anotadores son dos de los bloques de construcción básicos, especificados en la arquitectura.

Todos los componentes especificados en UIMA, poseen dos partes necesarias para su implementación:

1. La parte declarativa.
2. La parte de código.

La parte declarativa contiene metadatos que describen el componente, su identidad, la estructura y el comportamiento y son llamados "Component Descriptor". Los Component Descriptor están representados en XML. La parte de código implementa el algoritmo. La parte del código puede ser un programa en Java.

De esta forma, el comportamiento y estructura de los componentes UIMA, está dado por una serie de archivos XML, que pueden ser leidos por otras aplicaciones para determinar los resultados esperados.

3.1.5. USAR EL FRAMEWORK DESDE UNA APLICACIÓN.

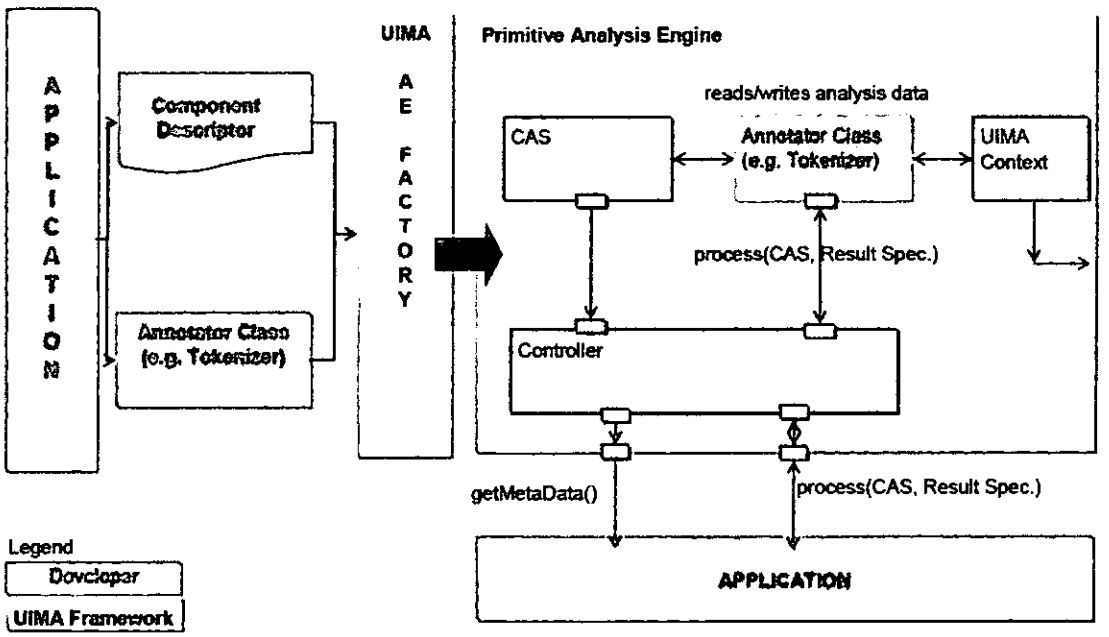


Ilustración 5: Usando el Framework UIMA, para crear e interactuar con un motor de análisis (Overview and Setup, 2009)

En la Ilustración 5, se puede observar en azul cada uno de los componentes de la arquitectura UIMA, y amarillo aquellos componentes que deberán ser proporcionados por el desarrollador. El proceso de interacción entre la arquitectura y la aplicación inicia cuando esta última instancia un AE, crea o adquiere una entrada CAS, inicializa la entrada CAS con un documento y luego se lo pasa al AE a través del método de process.

El UIMA AE Factory toma la información declarativa del Component

Descriptor y los archivos de clase en donde están las implementaciones de los anotadores, e instancia el AE instanciado por la aplicación, crea los CAS y los UIMA Context.

La aplicación puede acceder al proceso de análisis en cualquier momento, debido que este puede interactuar con diversos métodos de los CAS, que contienen toda esta información.

3.2. INVESTIGACIONES ANTERIORES.

El hecho de que la plataforma y el framework UIMA sea de libre uso y de licencia Apache (Apache Licence, 2011), ayuda de gran manera a que todos los aportes que se le hagan, estén disponibles para beneficiar a todos los desarrolladores interesados, además existen sitios en la red en donde se pueden encontrar todo tipo de componentes adaptables y funcionales (Carnegie Mellon University, 2010; Jena University, 2010), que pueden ser descargados gratuitamente y utilizados en aplicativos propios, como es el caso de motores de análisis para el procesamientos del lenguaje natural, o anotadores de textos.

Antes de elegir a UIMA como plataforma base para la construcción del

middleware, se estudiaron varias opciones:

1. La plataforma GATE (GATE, 2010), desarrollada en la Universidad de Sheeld desde 1995. Esta plataforma ha sido usada en una amplia variedad de proyectos de investigación y desarrollo del área de Procesamiento del lenguaje natural (PLN). GATE implementa la arquitectura mediante un framework y además provee un entorno de desarrollo construido sobre éste.

2. La plataforma Ellogon (Ellogon, 2010), 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.

3. Además, se estudiaron sistemas más recientes como LinguaStream (LinguaStream, 2010), en continuo desarrollo desde el año 2001.

El principal aspecto a tener en cuenta para la elección de la plataforma es el formato de representación del resultado del análisis, debido a que la interfaz que se pretende construir en este proyecto debe interactuar principalmente con estos datos, y estos deben poseer una estructura manejable y fácilmente interpretable, para conseguir los resultados esperados.

Al estudiar las plataformas se encontró que 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. Estas jerarquías de capas añaden un nivel más de complejidad al momento de interactuar con el resultado del análisis.

Por otra parte, una característica importante de UIMA es que puede trabajar con varios formatos como pueden ser audio, video, texto simultáneamente. 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 posee 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 (Techera, 2007).

En adición OASIS (Organization for the Advancement of Structured Information), instaló un comité técnico para la estandarización de la especificación de UIMA, en respuesta del actual crecimiento de investigaciones en institutos y empresas orientados al PLN, utilizando UIMA (Hahn et al. 2008).

Además de todo lo anterior, se eligió UIMA por su facilidad de desarrollo,

debido a que tiene una API con toda la información que se necesita con respecto a funcionalidad, por ser la única que contempla el procesamiento de videos y por estar sustentada por una amplia comunidad (IBM, 2010).

En la actualidad existen varios campos de estudio realizando investigaciones sustentadas y dirigidas hacia la plataforma UIMA, entre los cuales sobresalen:

- Lingüística Computacional.
- Medicina
- Biología

En el área médica nos encontramos con investigaciones orientadas a resolver problemas relacionados con la extracción de información biomédica, ayudando a los profesionales del área de la salud a realizar su trabajo y no perder tiempo en la digitación de información, al utilizar UIMA para procesar datos suministrados en lenguaje natural por los pacientes y luego almacenarlos en bases de conocimiento (Baumgartner et al., 2008). Actualmente existen varios componentes de UIMA para este proceso, como es el caso de BioNLP (BioNLP, 2010), alojado para ser descargado gratuitamente en Sourceforge.net

En otro caso la inteligencia artificial y la medicina se unieron, para trabajar sobre software y metodologías para la creación de sistemas expertos sobre procesos biomedicos, orientados por información no estructurada basados en UIMA (Mohamed Radhouene, 2008), facilitando su proceso de

automatización.

En el área de la biología sobresalen investigaciones realizadas en el área de microarrays[2], basados en UIMA (Wei-Bang, 2007). Demostrando la capacidad de procesamiento distribuido de la plataforma.

También se han creado software para la extracción de información relacionada con la interacción entre proteínas, utilizando la minería de textos sustentada en la plataforma, para que de manera inteligente y autónoma el sistema analice documentos de investigaciones científicas, y genere conocimiento relacionado con el tema (Rune Saetre et al., 2010).

La lingüística computacional es el área que más investigaciones ha realizado sobre UIMA, en la actualidad nos encontramos con componentes re-utilizables como motores de análisis, tokenizadores, typesystems, CAS, Collection Readers, y todo tipo de módulos que sustentan los procesos de PLN, generadores automáticos de resúmenes, herramienta para la traducción entre otros (Jena University, 2010; Hahn et al., 2008; Techera, 2007).

En este punto es relevante resaltar que el número de investigaciones importantes realizadas en países hispano-hablantes es muy inferior al realizado en Estados Unidos y Europa, limitando el número de componentes para el PLN en español. Entre los pocos trabajos se destacan "Lavinia" de Uruguay (Techera, 2007), una interfaz web para la integración de módulos

2 Herramienta para el procesamiento simultaneo de información de genomas

referentes al procesamiento del lenguaje natural y "WebQA" de Argentina, (Castelo et al., 2007) un sistema de búsqueda de respuestas para el idioma español utilizando la Web como repositorio de información .

El área de la interacción persona-ordenador, a pesar de que sus principales metas están relacionadas con el procesamiento de la información no estructurada (Diaper & Neville A, 2004), no ha recibido grandes aportes de la plataforma UIMA.

En forma de conclusión del capítulo, se tiene que para facilitar la utilización de datos no estructurados como interfaz de operación en aplicativos de cualquier tipo, se han realizados los siguientes adelantos:

1. Existen arquitecturas y frameworks que facilitan el procesamientos de datos no estructurados.
2. Hay disponible en la web gran cantidad de componentes reutilizables, gratis y de código abierto.
3. Existen entornos de programación que facilitan el proceso de interacción con dichas arquitecturas (Apache UIMA Development Community, 2009).

Sin embargo, hacen falta los siguientes aspectos para garantizar una verdadera facilidad en el desarrollo:

1. Existe un considerado nivel de complejidad al momento integrar la plataforma a una aplicación.
2. Las plataformas fueron diseñadas para expertos en el área del procesamiento de información no estructurada.
3. Hay que conocer de antemano, los posibles resultados del procesamiento de las plataformas. Información que le dificulta el trabajo a un desarrollador estándar.

4 OBJETIVOS Y ALCANCE.

4.1 OBJETIVO GENERAL:

Construir un middleware, que funcione como interfaz de comunicación hacia UIMA, que facilite a constructores de software, implementar en sus aplicativos la opción de operarlos mediante datos no estructurados, procesados en tiempo de ejecución por la plataforma UIMA.

4.2 OBJETIVOS ESPECÍFICOS:

i. Proponer un modelo conceptual para diseñar y desarrollar aplicativos fácilmente integrables a la arquitectura UIMA, con el fin de utilizar el poder de análisis de información no estructurada que ésta arquitectura posee.

ii. Estudiar el área del procesamiento de datos no estructurados y la representación de su análisis, investigando cómo acceder a esta última para transformarla en una estructura de datos manejable.

iii. Encontrar la forma para que solo fragmentos del resultado del análisis de la información que encajen con un determinado patrón, sean quienes

desencadenen las acciones pertinentes en la aplicación.

iv. Implementar un perfil operativo del modelo conceptual, que permita validarlo y evaluar su acoplamiento a cualquier aplicativo.

5 METODOLOGÍA

Al realizar la revisión literaria, se reveló que únicamente hay guías no investigadas e ideas vagamente relacionadas con el tema de investigación, lo que conllevó a que se realizara un estudio exploratorio de éste, y que se procediera a una investigación de tipo experimental.

Este proyecto posee un diseño de investigación experimental, ya que se evaluaron y manipularon diferentes variables (tecnologías, patrones y herramientas), para construir la herramienta que facilitara de la mejor forma el proceso de construcción de software, con la opción de operarlos mediante datos no estructurados, procesados en tiempo de ejecución por la plataforma UIMA.

La investigación se realizó en las instalaciones de la Universidad de Cartagena y tuvo una duración de 6 meses, iniciando desde el 30 de Agosto de 2010.

5.1 PROCEDIMIENTOS

El primer paso en la investigación, fue realizar una documentación y revisión del estado del arte del procesamiento de datos no estructurados y la

representación de su análisis en la plataforma UIMA. En donde se hizo una búsqueda exhaustiva de la documentación representada en : tutoriales, API's, artículos científicos, proyectos de investigación y manuales referentes a las plataformas de procesamiento de información no estructurada y el funcionamiento especifico en UIMA (Lo anterior contribuye al alcance del objetivo específico No. 2).

Luego de conocer la forma en que UIMA estructura el resultado del análisis y lo dispone a la aplicación que lo solicita, se analizó los requerimientos ingenieriles y tecnológicos necesarios para establecer una interfaz de operación entre la plataforma y la aplicación cliente. En este paso se realizó un levantamiento de análisis funcionales y no funcionales.

Paso seguido se realizó un estudio y evaluación de patrones de diseño y arquitectónicos, que dieran soporte a la plataforma. Además se realizó una búsqueda de entornos de desarrollo, lenguajes de programación y herramientas que facilitaran el proceso de construcción de la interfaz (Lo anterior contribuye al alcance del objetivo específico No. 1).

Luego de tener los requerimientos del software y las herramientas tecnológicas disponibles, se diseñó la arquitectura de la interfaz. Posteriormente se hizo un diseño de la micro-arquitectura y la documentación pertinente (Lo anterior contribuye al alcance del objetivo específico No. 3).

Una vez finalizada la etapa de diseño, se continuó con la implementación de la interfaz. En esta etapa se culminó un componete independiente, listo para entre-conectar UIMA y una aplicación cliente.

Para comprobar en qué grado facilitaba la interfaz elaborada, al proceso de construcción de software que utilicen datos no estructurado como interfaz de operación utilizando UIMA, se desarrolló un prototipo funcional o ejemplo de la utilización, en donde se prueba la efectividad del proceso. Se documentó la implementación de este y se realizaron las correcciones pertinentes a la interfaz (Lo anterior contribuye al alcance del objetivo específico No 4).

5.2 RECOLECCIÓN DE INFORMACIÓN.

Debido a que la investigación intentaba buscar una forma de "facilitar" el proceso de construcción de software con las características de operación antes mencionadas, la manera de recolectar información es documentando en qué medida se facilitó dicho proceso.

En este documento se hace un listado de los pasos que se siguieron, para culminar la construcción del prototipo de prueba, en donde se demuestra la facilidad del proceso.

Para medir el rendimiento de la interfaz, se realizaron varias pruebas de ejecución y se calcularon los tiempos de procesamiento de ésta. Dichas pruebas fueron realizadas con la herramienta Profiler de Netbeans (Netbeans Profiler, 2011).

5.3 CONSTRUCCIÓN DEL MIDDLEWARE

Para que los constructores de software pudieran implementar aplicaciones operadas mediante datos no estructurados, de forma fácil y sin la necesidad de poseer grandes conocimientos en el área del procesamiento de información, se hizo necesario crear una interfaz capaz de interactuar con la arquitectura UIMA, y acceder a su poder de procesamiento, para ponerla a disposición de la aplicación del constructor.

A continuación se muestra la documentación del proceso de diseño e implementación de la interfaz.

5.4 REQUISITOS DEL SOFTWARE

La principal función de la interfaz, es tomar información no estructurada y enviarla a UIMA para que esta la procese y luego dependiendo del resultado de ese análisis operar la aplicación. Es por esto que la interfaz posee un solo caso de uso que es "operar aplicación" y un solo actor que es "la aplicación".

5.4.1 RESTRICCIONES

Listado de Restricciones	
Id	*Descripción*
RES-1	La solución deberá ser un componente reutilizable (librería).
RES-2	La solución deberá ser implementada en JAVA.

Tabla 1: Listado de restricciones del software

5.4.2 REQUERIMIENTOS FUNCIONALES

Listado de Requerimientos Funcionales	
Id	*Descripción*
RQF-1	El sistema debe ser capaz de conectarse con la arquitectura UIMA y utilizar su poder de procesamiento de información no estructurada
RQF-2	El sistema deberá poder conectarse con cualquier aplicación construida en JAVA, y hacer llamado de sus métodos en el momento adecuado.
RQF-3	El sistema debe funcionar como puente, entre UIMA y la aplicación del usuario, llevando las entradas de la aplicación (texto, audio o video) a UIMA para que este la procese y con este resultado poder operar la aplicación cliente.

Tabla 2: Requerimientos funcionales.

5.4.3 REQUERIMIENTOS NO FUNCIONALES

Listado de Requerimientos del Sistema	
Id	*Descripción*
RQS-1	El sistema deberá adaptarse y configurarse de forma fácil a la aplicación cliente, de tal forma que no sea necesario hacer grandes modificaciones al código fuente.
RQS-2	El sistema deberá estructurarse de tal manera que sea posible en el futuro adaptarse a nuevas versiones de UIMA.
RQS-3	Debe ser capaz de soportar grandes cantidades de información, sin que ningún fragmento de ésta se pierda.

Tabla 3: Requerimientos no funcionales.

5.5 ANÁLISIS Y DISEÑO

Para dar soporte a los anteriores requerimientos, se propone la construcción de una interfaz que funcione como puente entre UIMA y la aplicación cliente. La interfaz podrá llamar diferentes métodos de clases de la aplicación, y así "operarla", ejecutando solo los métodos que el constructor de software indique mediante un archivo externo y cada método estará vinculado a un evento. Por otra parte, la interfaz enviará a UIMA la información no estructurada recogida por la aplicación cliente, y luego del procesamiento, tomará el resultado del análisis y lo comparará con cada uno de los eventos relacionados con los métodos, y si alguno de estos coincide se procederá a su ejecución.

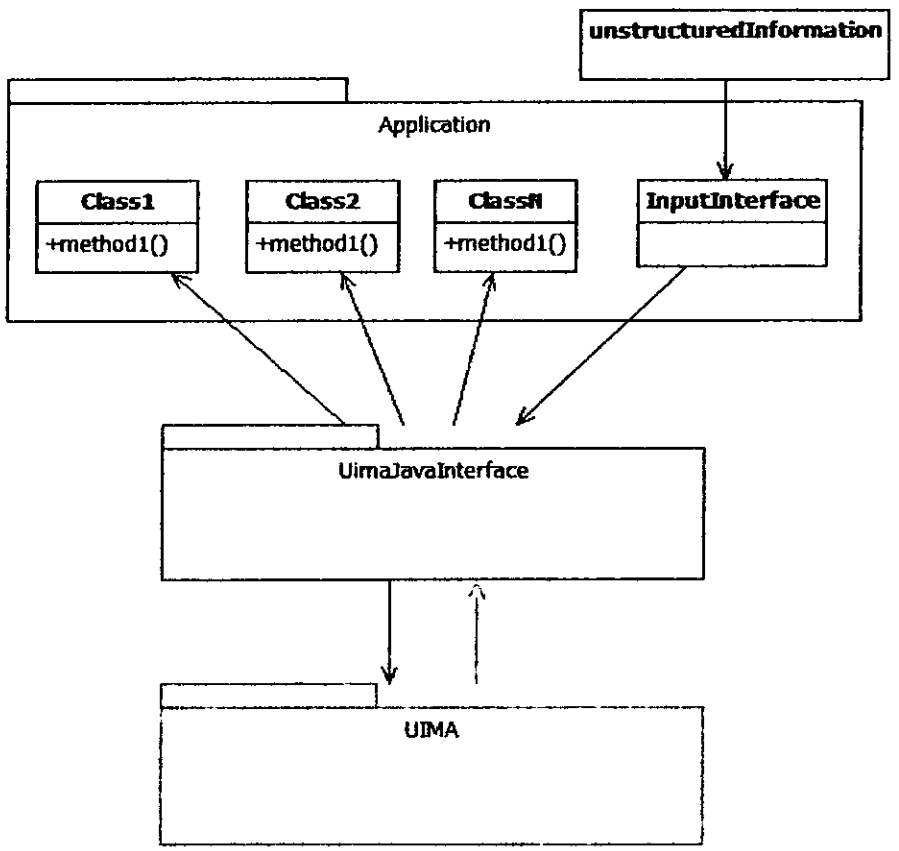


Ilustración 6: Funcionamiento de la interfaz

En la Ilustración 6 se muestra que la aplicación cliente posee una serie de clases, que son la base se su funcionamiento y que el llamado a los métodos de cada una de estas conforman el comportamiento de la aplicación. Además esta posee una interfaz de entrada, que es la encargada de recibir el flujo de datos que la controlaran. El flujo de información no estructurada es enviado a la interfaz, en este punto denominada UimaJavaInterface (UJI) y esta la envía hacia el componente UIMA para su procesamiento. Finalmente la interfaz

accede al resultado del análisis y dependiendo de éste realiza los respectivos llamados a los diferentes métodos de la aplicación.

A continuación se muestra de forma general, la manera en que los diferentes componentes interactúan (aplicación cliente, UIMA e interface).

5.5.1 INTERACCIÓN CON LA APLICACIÓN

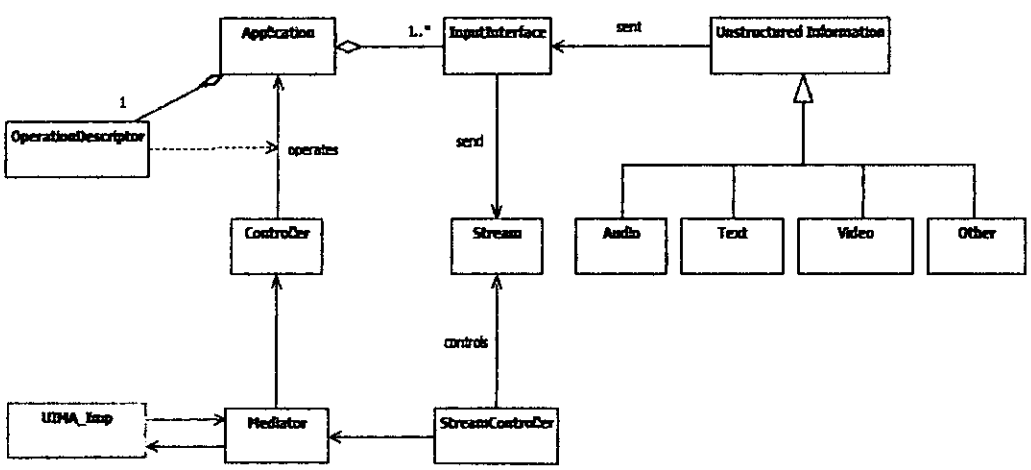


Ilustración 7: Interacción entre componentes

En la Ilustración 7 se observa que la aplicación cliente deberá poseer un operationDescriptor, que es un artefacto externo que contiene la información de cada uno de los métodos de las diferentes clases de la aplicación que

podrán ser ejecutados en el proceso. Además tendrá un inputInterface, que es el componente de la aplicación encargado de recibir el flujo de información no estructurada, que podrá presentarse en formato de texto, audio o video principalmente. Estos flujos de datos de diferentes pesos y velocidades se enviaran directamente a la interfaz, quien por medio de un streamController se hará responsable de que cada fragmento de información sea procesada correctamente. Un elemento de la interfaz denominado mediator es el que se encarga de interactuar con la implementación de UIMA, enviando los flujos de datos para que sean procesados y posteriormente direccionando sus respectivos resultados del análisis al controller. Finalmente el controller recibe el resultado y lo compara con los diferentes eventos suministrados en el operationDescriptor y realiza los respectivos llamados a los procedimientos relacionados.

5.5.2 VISTA LÓGICA

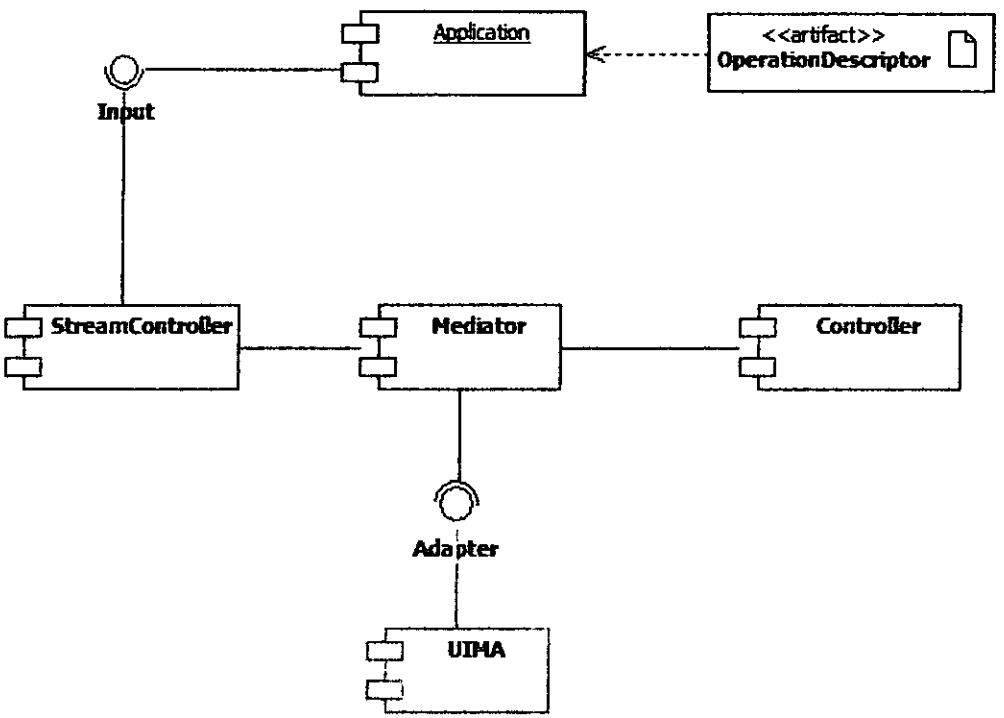


Ilustración 8: vista lógica

En la arquitectura de la interfaz se propusieron tres componentes principales:

I. StreamController: Se encarga de gestionar el flujo de información que entrará a la interfaz. Como las velocidades y pesos de los flujos de información son variantes, así como el tiempo que tarda la implementación de UIMA en procesarlos, este componente segmenta la información en paquetes que son transportados por toda la interfaz.

II. Mediator: Administra los paquetes enviados por el StreamController, de tal forma que si el componente UIMA se encuentra temporalmente ocupado procesando paquetes anteriores, este los almacena en una estructura de datos, para garantizar su posterior procesamiento. Además el componente aquí descrito enviá el resultado de la información al Controller.

III. Controller: Es el componente encargado de operar la aplicación cliente. Comparando el resultado del análisis con los diferentes eventos relacionados en el operationDescriptor, el controller tiene la facultad de identificar los métodos de las clases correspondientes y ejecutarlos en el acto.

5.5.3 VISTA DE DESARROLLO

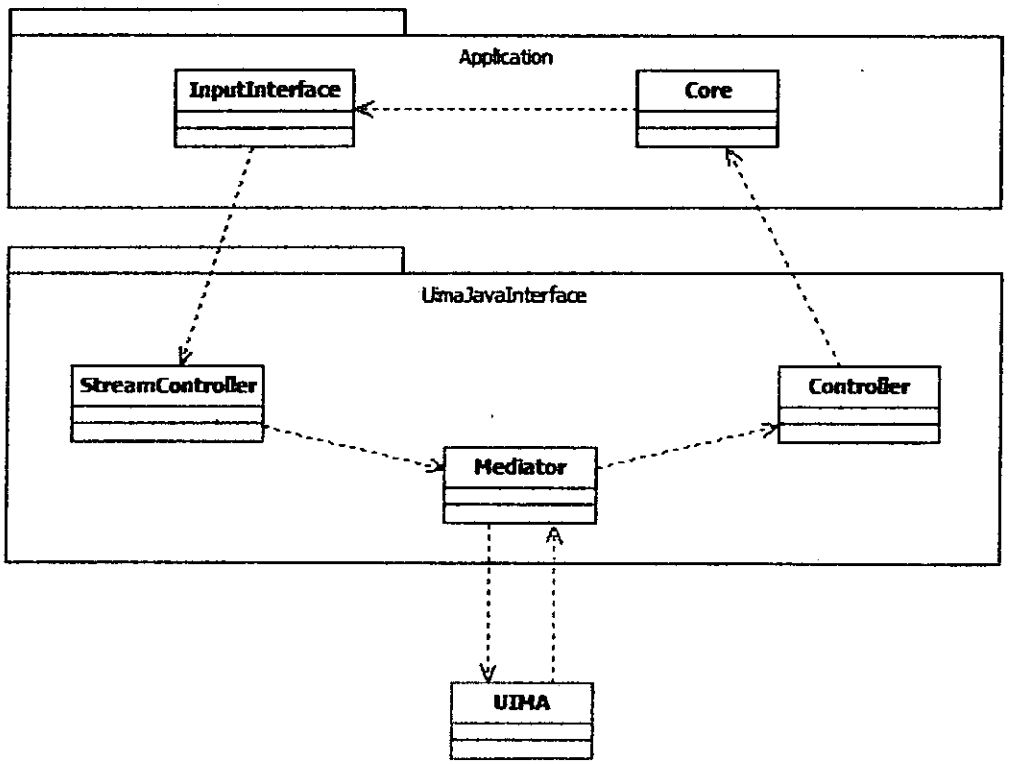


Ilustración 9: vista de desarrollo

En la vista arquitectónica de desarrollo Ilustración 9, se vislumbra que UimaJavaInterface funciona como un puente entre la aplicación cliente y el componente UIMA.

El componente *core,* representa todas aquellas clases y elementos que permiten el funcionamiento de la aplicación cliente. Además la aplicación

cliente contiene un componente *inputInterface,* que recibe la información no estructurada y la pasa al componente UJI, más precisamente al *streamController* que maneja los flujos de información, para luego utilizar el *mediador,* El encargado de intermediar con el componente UIMA, para finalmente pasarle el resultado del procesamiento al *controller,* quien ejecutará los diferentes métodos del *core* de la aplicación cliente, para así operarla.

5.5.4 MICRO-ARQUITECTURA

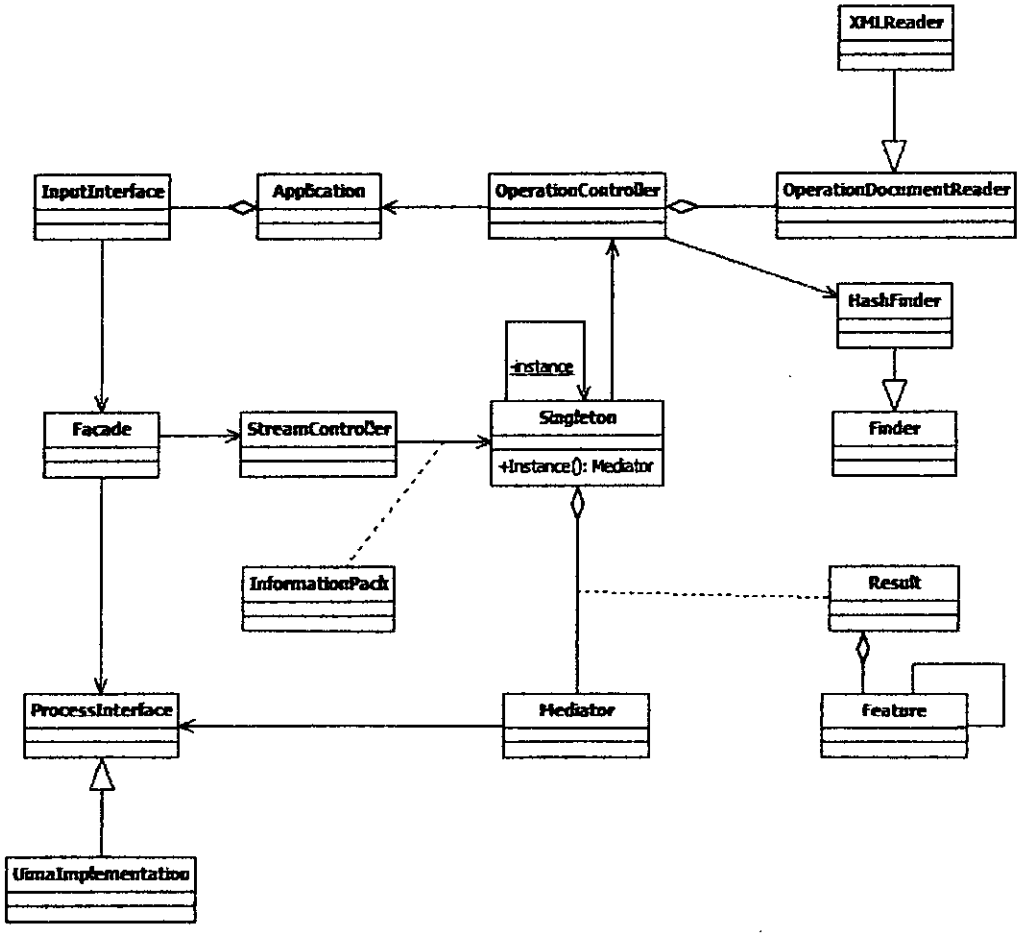


Ilustración 10: diagrama de clases

A continuación se explica la funcionalidad de cada una de las clases mostradas en la Ilustración 10:

A. Facade: Esta clase hace parte de un patrón de diseño que lleva su mismo nombre, que sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas (GoF, 1995). En este caso provee una forma simplificada de acceder a las funcionalidades de la interfaz desde la aplicación cliente.

B. StreamController: Esta clase recibe los diferentes flujos de información no estructurada y los fragmenta en diferentes paquetes para que puedan ser manejados de mejor forma. Además de empaquetar la información, envía todos los fragmentos al siguiente nivel.

C. InformationPack: Mediante esta clase se empaquetan los flujos de datos. Además del segmento de información, lleva consigo la instancia de la aplicación a controlar que llega hasta el final del proceso.

D. Singleton: Esta clase es el núcleo del patrón singleton, que consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella (GoF, 1995). En la interfaz se utiliza para proporcionar un solo flujo de información hacia el componente UIMA, garantizando que paquetes de información no se pierdan por la falta de

disponibilidad del componente. El singleton garantiza que el mediador solo posea una instancia dentro del proceso, lo que permite que todos los paquetes lleguen al mismo objeto y que sean procesados correctamente.

E. Mediator: Es la clase que coordina el procesamiento de la información. El mediador recibe cada uno de los paquetes enviados por el StreamController, y los apila en espera de su procesamiento, luego de enviarlos uno por uno los quita de la pila y recoge su resultado para enviarlo al siguiente nivel.

F. ProccesInterface: Es una clase abstracta que provee una interfaz de procesamiento a UJI. Las clases que hereden de ésta deberán implementar los métodos para el procesamiento de la información no estructurada. Inicialmente la única implementación concreta de ésta clase, será el componente de UIMA, pero de igual forma en el futuro se podrán utilizar otros procesadores de información no estructurada que no sean UIMA y anexarlos.

G. UimaImplementation: Como se menciono anteriormente, UimaImplementation es la implementación concreta de ProccesInterface y se encarga de ofrecer el contrato de procesamiento a la interfaz. Esta clase debe contener todo el código necesario para que UIMA funcione.

H. Result: Es la clase mediante la cual UJI envía el resultado del análisis a sus diferentes componentes. Consiste en una estructura de datos en forma de árbol que permite manejar la información estructurada. Cada objeto de Result representa el procesamiento de un fragmento de información, y a su vez es la raíz de ese árbol.

I. Feature: Se utiliza para almacenar cada una de las características del resultado del análisis. En la estructura de datos representan nodos hijos del result, y a su vez es posible que tengan sus propios hijos (otros features).

J. OperationDocumentReader: Es una clase abstracta que ofrece el contrato de lectura del operationDocument. Como este artefacto externo se puede encontrar en diferentes formatos, es necesario crear implementaciones que garanticen que la lectura de este sea transparente para UJI.

K. XMLReader: Es la implementación concreta de OperationDocumentReader, que lee el artefacto escrito en formato XML.

L. Finder: Es una clase abstracta que proporciona un contrato a UJI, para comparar el resultado del análisis con la información de los eventos

recopilada en el OperationDocument. Si al comparar se encuentran coincidencias de eventos en el resultado del análisis, se genera una lista de los métodos a ejecutar y se envía al OperationController.

M. HashFinder: Es una implementación del finder, que genera códigos únicos del resultado del análisis y otros de los eventos, y luego los compara y genera la lista de métodos.

N. OperationController: Esta clase recibe el resultado del análisis y a continuación lee el OperationDocument mediante el OperationDocumentReader, luego utiliza el Finder para realizar las comparaciones y luego de generar la lista de métodos procede mediante la instancia de la aplicación cliente a ejecutar cada uno de los procedimientos.

5.6 IMPLEMENTACIÓN

Como se diseño en la arquitectura, el sistema está compuesto por tres componentes principales, como se muestra en el siguiente diagrama de componentes.

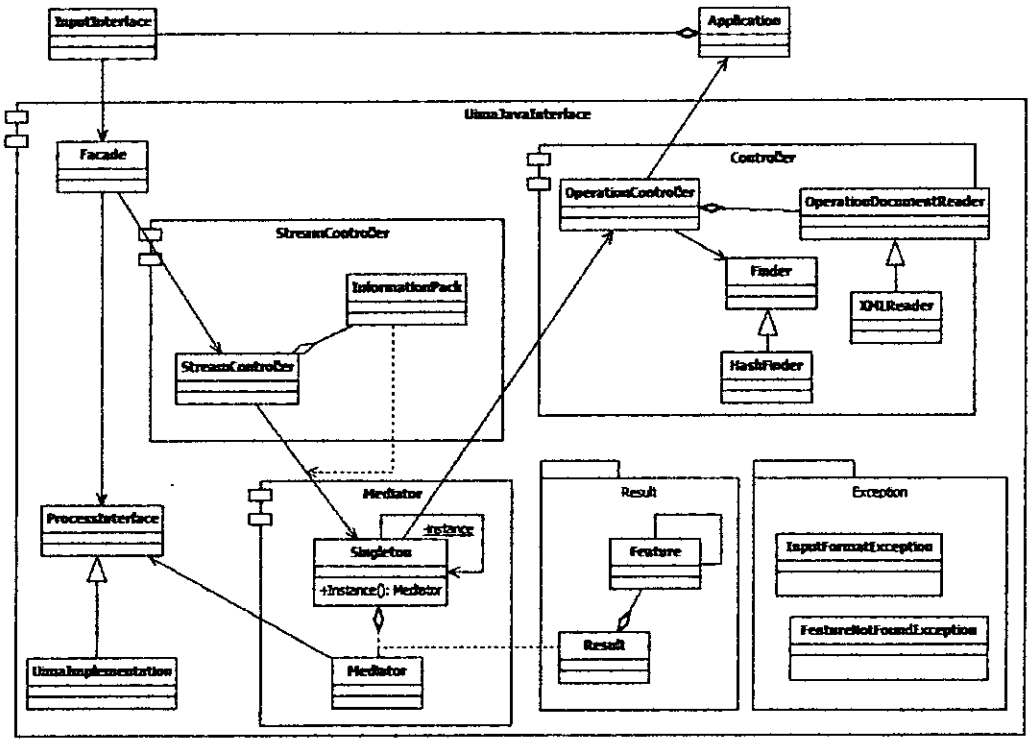
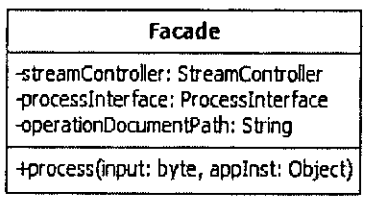


Ilustración 11: diagrama de componentes

A continuación se describe el funcionamiento en detalle de los principales elementos de UimaJavaInterface.:

- **Clase Facade**: A la fachada hay que configurarle la ruta de acceso al operationDocument para que pueda funcionar. Además se le envía una instancia de una implementación de ProcessInterface, que será el componente que procese la información no estructurada. La fachada internamente posee una referencia hacia el StreamController, para continuar el flujo de datos. Posee un único método público y estático que recibe como parámetro una objeto de la clase Object, con el flujo de

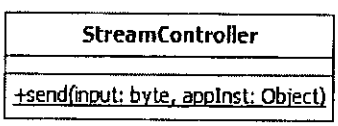
información a procesar y una instancia de la clase de la aplicación cliente a controlar, este es el método que inicia el proceso.



Clase 1: Facade

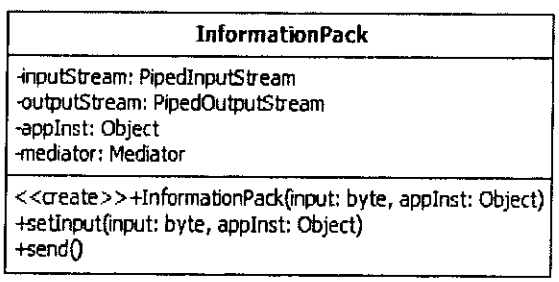
I. StreamController

a. **Clase StreamController**: Es una clase que contiene un único método publico y estático que recibe como parámetro una objeto de la clase Object, con el flujo de información a procesar, además es necesario enviarle la instancia de la clase de la aplicación cliente a controlar.



Clase 2: StreamController

b. **Clase InformationPack**: Aquí se utilizaron los atributos de las clases PipedInputStream y PipedOutputStream debido a que estas son convenientes para métodos que producen una salida que son utilizada como entrada de otro, es decir se utilizan como tubería para interconectar los componentes de UJI. Además estas clases soportan la sincronización de hilos, lo que las hace adecuadas ya que algunos componentes trabajan mediante threads. Además de llevar la instancia de la aplicación cliente, este tiene una referencia hacia el mediador. Mediante el método setInput se empaqueta la información no estructurada enviada por la aplicación y se introduce en la entrada de la tubería (inputStream), Conectando y enviando el paquete hacia el mediador.

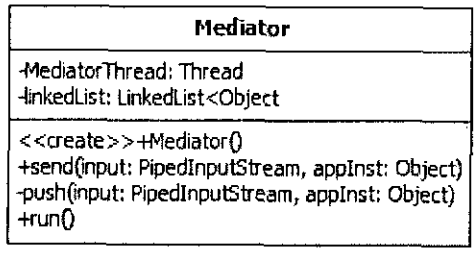


Clase 3: InformationPack

II. Mediator:

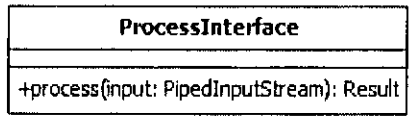
a. **Clase Mediator:** Es la clase que contiene un hilo de procesamiento, por el cual envía información a UIMA y espera su resultado, ya que

mientras la arquitectura procesa la información, la interfaz puede ir recibiendo más datos. El mediador acumula los paquetes mediante el atributo linkedList, que es una lista enlazada implementada como pila. Los datos entran mediante el método publico send y utiliza push para ir almacenando los paquetes. El primer elemento de la pila será enviado a procesar a la implementación del método run, y de aquí partirá al siguiente nivel, y el siguiente elemento de la pila realizará el mismo recorrido hasta acabar con la pila.



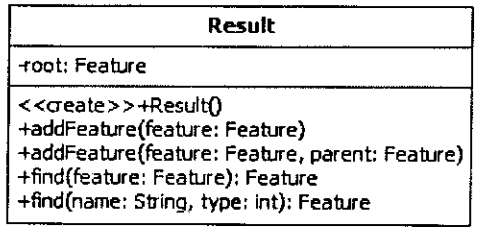
Clase 4: Mediator

- **Clase ProcessInterface:** Provee un método abstracto, para el procesamiento de la información no estructurada. La clase que lo implementa deberá construir el árbol mediante Result y retornarlo como resultado del análisis.



Clase 5: ProcessInterface

- **Clase Result:** Contiene el árbol generado por el processInterface. En el atributo root se almacena el feature que será la raíz de la estructura. Contiene varios métodos para facilitar el almacenamiento de la información.

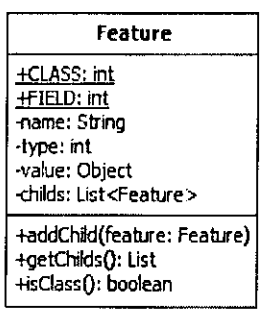


Clase 6: Result

- **Clase Feature:** Aquí se representan las características encontradas en la información no estructurada. Se le puede asignar un nombre mediante name y el valor se guarda en value, además como una característica puede contener a otras se debe especificar mediante el atributo type, que es un entero que puede tomar los siguiente valores:

1. Si la característica contiene a otras (En UJI se le denomina CLASS).
2. Si la característica no contiene a otras (En UJI se le denomina FIELD).

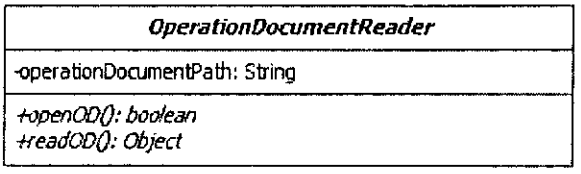
Además esta posee métodos para facilitar la construcción de la estructura.



Clase 7: Feature

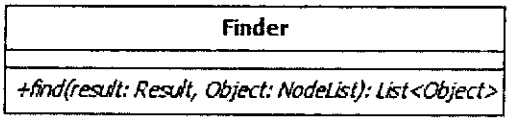
III. Controller:

a. **Clase OperationDocumentReader:** Esta clase abstracta contiene un atributo en donde se almacena una cadena con la ruta en donde se encuentra el operationReader. Esta contiene dos métodos públicos y abstractos para la apertura y lectura del documento.



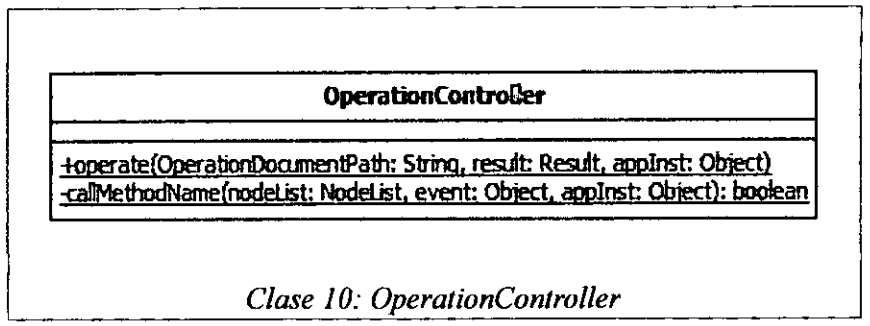
Clase 8: OperationDocumentReader

b. **Clase Finder:** Es una interface que provee un método a implementar, que recibe por un lado el resultado del análisis y por otro la estructura leída del operationDocument. Su implementación debe retornar una lista con los nombres de los métodos a ejecutar.



Clase 9: Finder

c. **Clase OperationController:** Mediante el método operate se crean instancias de OperationDocumentReader y Finder, para leer el artefacto y comparar con el resultado respectivamente. Luego de obtener la lista de métodos se llama a callMethodName para ejecutarlas una por una.



Clase 10: OperationController

Luego de conocer en detalle el funcionamiento de cada clase, se facilita la comprensión del funcionamiento global de la interfaz. A continuación se muestra el diagrama de secuencia para visualizar todo el proceso.

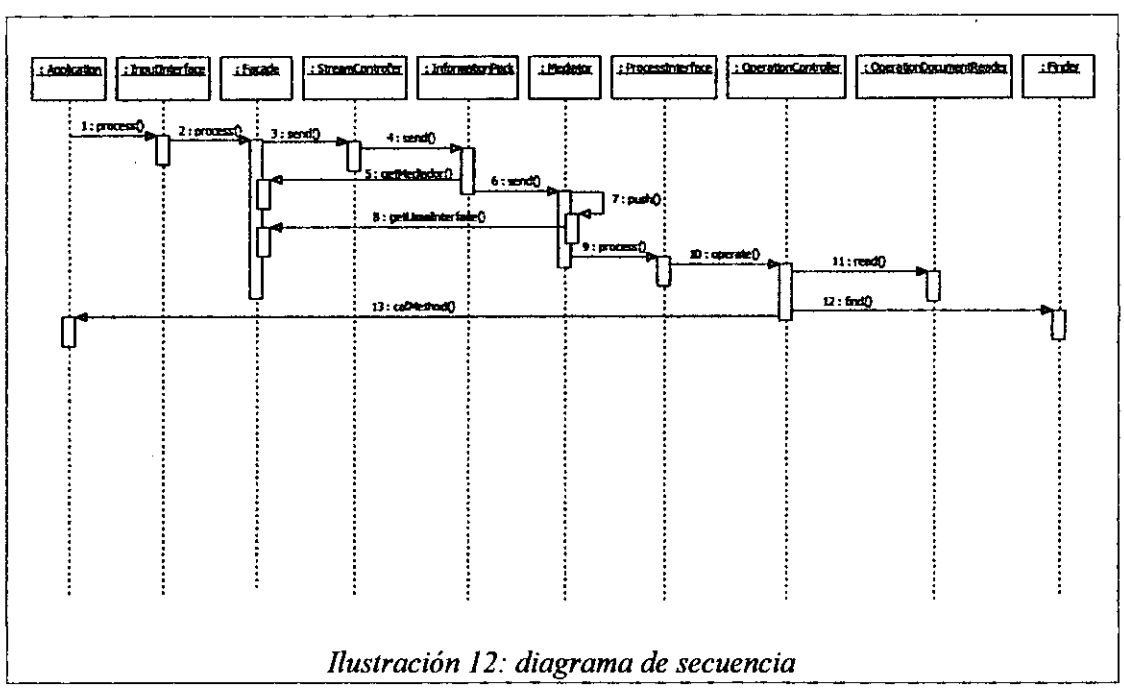


Ilustración 12: diagrama de secuencia

Para finalizar el proceso, es necesario comprender la estructura que se diseñó para el operationDocument en su implementación en XML.

El documento contiene dos elementos principales, que son los eventos y las clases a operar. Cada evento consta de un tipo y una serie de características, en donde el tipo representa las entidades que se pueden encontrar en la información, como por ejemplo una palabra encontrada en un texto o una persona en una imagen; y las características son aquellos atributos del tipo. Cada tipo posee un identificador único dentro del documento y un nombre, mientras que las características poseen un nombre y el valor encontrado. Si al tipo se le especifica el atributo *strict* en true, el sistema deberá encontrar únicamente todas las características especificadas para considerar que el evento ocurrió.

Por otra parte a cada clase hay que especificarle el nombre y por lo menos un método a ejecutar. Cada método posee un nombre y una serie de eventos a los cuales este reaccionará. Las clases se relacionan con los eventos mediante el identificador. A un método se le puede asignar uno o más parámetros de entrada, de la lista de información que activó el evento, un ejemplo de esto podría ser que el evento sea "un nombre seguido de un verbo" , y en el método relacionado le especifique que va a recibir un parámetro, al momento de que UJI encuentre "Daniel escribe su tesis" y active el evento, al método asignado le pasará "Daniel" como parámetro. Para especificar de qué características se tomarán los parámetros se utiliza *param-source* y asignarle el nombre de la

ésta, además de especificar mediante *param-token* cuantos parámetros se pasaran, o el comodín *ALL*, si se quieren todos.

A continuación se muestra un ejemplo de operationDocument:

```
<root>
<events>
        <type name="Tag" id="1" strict="true">
                <feature name="tag">VN</feature>
                <feature name="word">activar</feature>
        </type>
        <type name="Tag" id="2" strict="true">
                <feature name="tag">VN</feature>
                <feature name="word">leer</feature>
        </type>
        <type name="Tag" id="3" strict="true">
                <feature name="tag">VN</feature>
                <feature name="word">regresar</feature>
        </type>
        <type name="Tag" id="4" strict="true">
                <feature name="tag">VN</feature>
                <feature name="word">terminar</feature>
        </type>
        <type name="Tag" id="5" strict="true">
                <feature name="tag">NC</feature>
                <feature name="word">ayuda</feature>
        </type>
```

```
        <type name="Tag" id="6" strict="true">
                <feature name="tag">NC</feature>
                <feature name="word">proxima</feature>
        </type>
        <type name="Tag" id="7" strict="true">
                <feature name="tag">AQ</feature>
                <feature name="word">anterior</feature>
        </type>
</events>
<classes>
        <class name="Main">
                <method name="activar" param-source="word" param-token="ALL">
                        <event id="1"/>
                </method>
                <method name="leerCuento" param-source="word" param-token="ALL">
                        <event id="2"/>
                </method>
                <method name="principal">
                        <event id="3"/>
                </method>
                <method name="cerrarVentana">
                        <event id="4"/>
                </method>
                <method name="ayuda">
                        <event id="5"/>
                </method>
                <method name="siguiente">
                        <event id="6"/>
```

```
            </method>
            <method name="anterior">
                <event id="7"/>
            </method>
        </class>
</classes>
</root>
```

6 RESULTADOS.

Como producto de la investigación, se obtuvo un middleware denominado UJI (UIMA Java Interface), representada en una librería funcional para Java (un archivo jar), completamente finalizado y probado, listo para su utilización en cualquier aplicativo construido en java.

En la siguiente tabla se describen cada uno de los pasos y conocimientos necesarios para anexar a un aplicativo la opción de operarla mediante datos no estructurados.

Orden	Paso	Conocimientos previos
1	Descargar e importar cualquier componente de procesamiento de UIMA[3], cuya único requerimiento es que tenga un método publico que devuelva un objeto de tipo "JCAS" (Apache UIMA Development Community, 2009).	Programación básica.
2	Importar la librería UJI	Manejo básico de IDE.
3	Crear una clase que implemente "ProcessInterface" de UJI, y en el método "procces", tomar el objeto retornado por el componente UIMA y enviárselo al constructor de un objeto de "UimaProcessInterface" y luego devolver lo retornado por el método "process" de este último objeto.	Programación Orientada a objetos, programación en Java.
4	Crear y configurar el archivo "OperationDocument.xml", con los eventos y métodos correspondientes.	Manejo de XML.
5	Obtener una instancia de la clase "Facade" de UJI y enviarle una instancia de la implentación de "ProcessInterface" y la ruta del archivo "OperationDocument.xml".	Programación en Java.

3 En este documento se sugieren varios repositorios , sin embargo en la red se pueden encontrar muchos más.

6	Enviarle los datos no estructurados a la interfaz.	Programación básica.

Tabla 4: Pasos para utilizar UJI.

A partir de la tabla 4, es posible afirmar que cada paso es considerablemente sencillo, y que virtualmente cualquier constructor de software sin la necesidad de ningún conocimiento en el área del procesamiento de datos no estructurados, podría ejecutarlo sin ningún problema.

Para una mayor comprensión de la configuración y utilización de UJI se creó un manual del sistema que lo documenta de una forma clara y sencilla. Además existe un API de la librería en formato html para tener una referencia más precisa.

Para probar la facilidad de implementación de la interfaz, se construyó un software para la lectura de cuentos, con la característica de operación mediante el lenguaje natural, utilizando la voz.

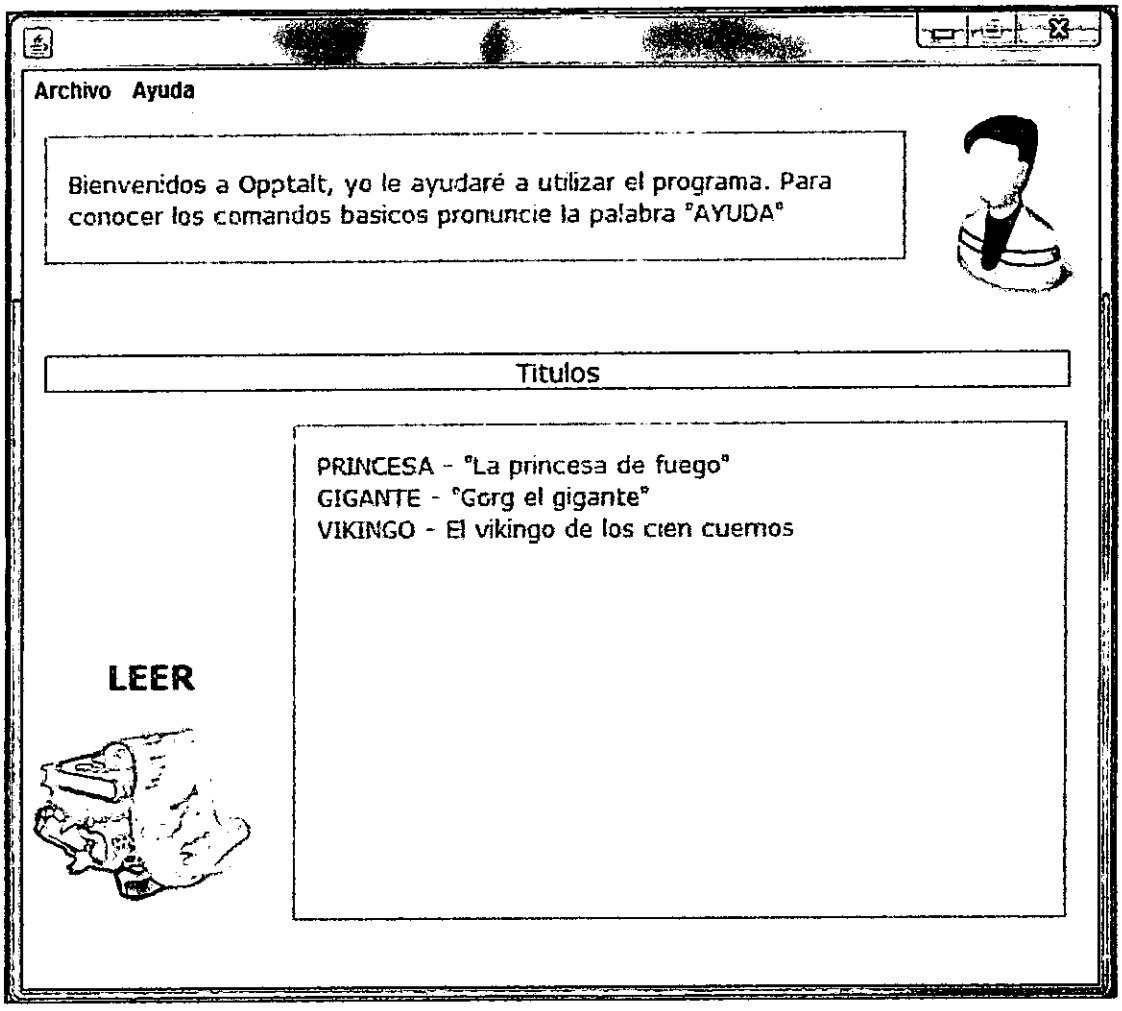


Ilustración 13: Ventana principal (Opptalt)

El prototipo fue llamado Opptalt, que en noruego significa "contado". El software le muestra al usuario una lista de títulos de cuentos disponibles, cada cuento se encuentra almacenado en un archivo de texto y en el listado se muestra el nombre del archivo, seguido por el título, como se muestra en la Ilustración 13.

En este prototipo se utilizó el componente de procesamiento OpenNPL para anexarlo a UJI y poder operar el aplicativo mediante el lenguaje natural. Más concretamente se utilizó un tokenizador, pos tagger y un sentence detector.

Para seleccionar el cuento a leer solo basta con pronunciar la palabra "leer", seguido del nombre del archivo que aparece en la lista. Ej: leer princesa. Al pronunciar estas palabras el sistema envía los datos no estructurados al componente de procesamiento, luego UJI interpreta el resultado del análisis y ejecuta la orden correspondiente, que es ocultar la ventana principal y abrir la ventana de lectura Ilustración 14.

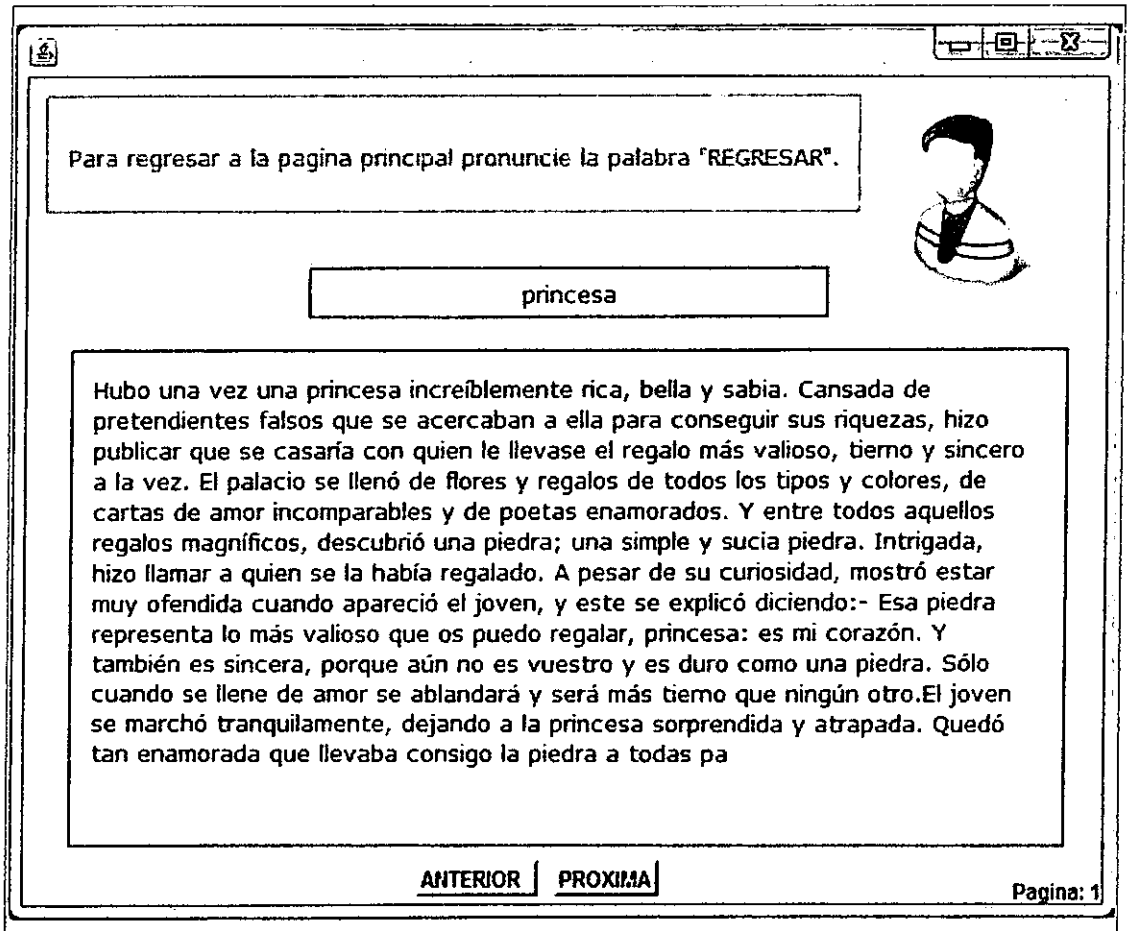


Ilustración 14: Ventana de lectura (Opptalt)

Para regresar a la página anterior o avanzar a la próxima, se deben pronunciar las palabras "anterior" y "próxima" respectivamente. Para regresar a la ventana principal solo basta con pronunciar "regresar", o para abrir la ventana de ayuda "ayuda". En la ventana de ayuda se listan los comandos y combinaciones de palabras para realizar diversas acciones Ilustración 15.

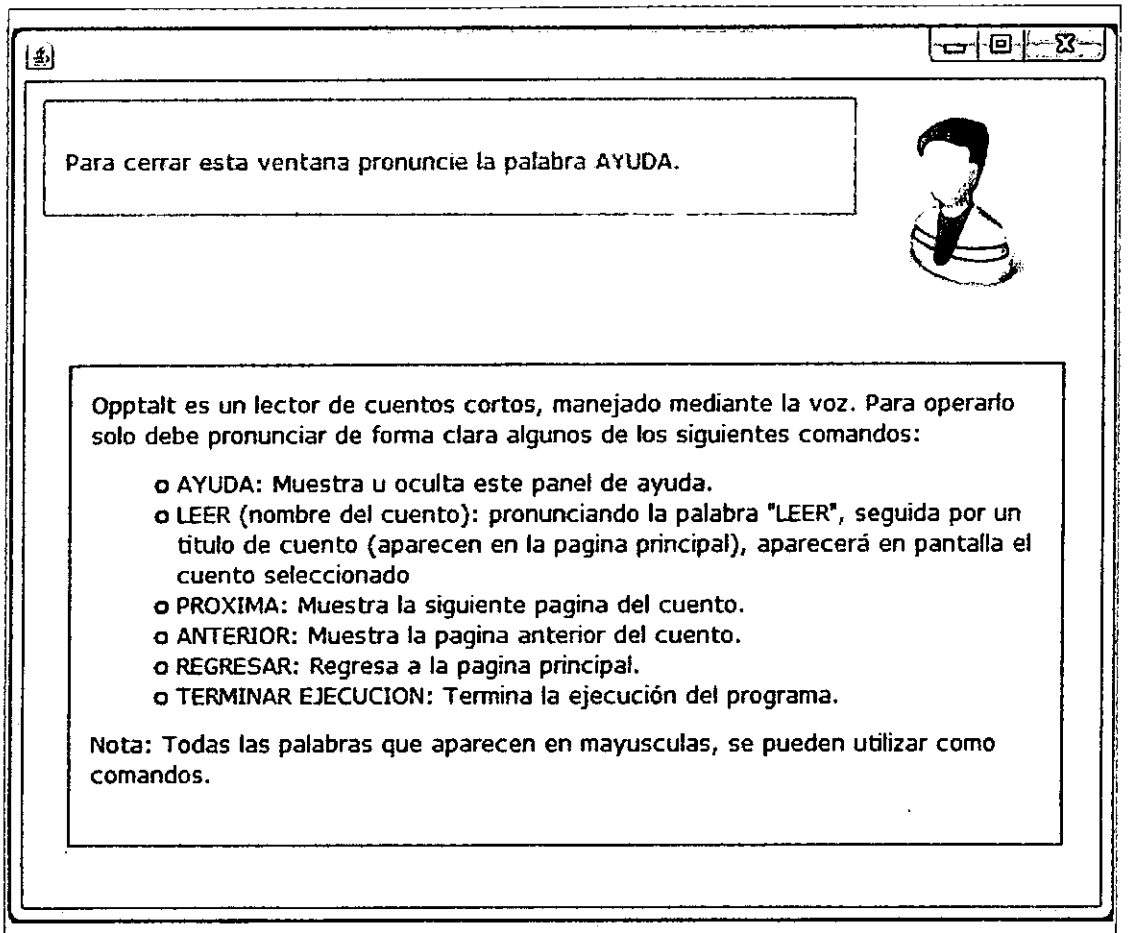


Ilustración 15: Ventana de ayuda (Opptalt)

En cualquier momento se puede pronunciar la frase "Terminar Ejecución" y el software se cerrará.

Por otra parte se implementó un pequeño juego operado mediante texto en lenguaje natural,
donde existen 4 personajes representados por redes sociales, a las cuales hay que guiar para encontrar a un usuario Ilustración 16.

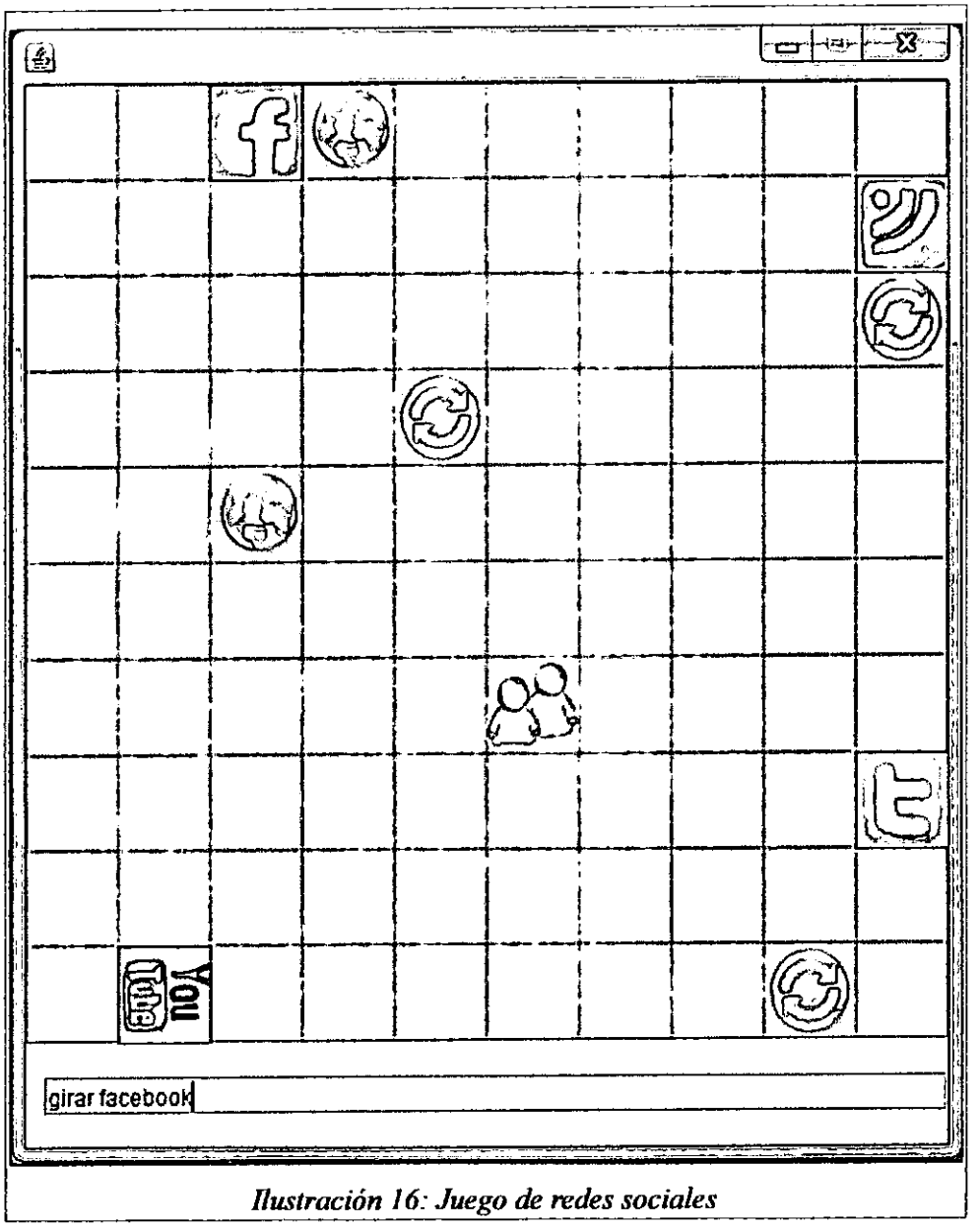


Ilustración 16: Juego de redes sociales

En este juego dependiendo de la entrada de texto, los personajes irán avanzando y girando todos a la vez o uno por uno, obedeciendo al tipo de

instrucción. Además existen ítems que modificaran su comportamiento. El fin del juego es alcanzar la casilla del usuario sin que 2 o más redes sociales choquen entre sí. Como es de notar este es un simple juego que fácilmente pudo ser implementado para operarlo mediando botones o por el mouse, pero sirvió para demostrar que independientemente del ámbito del aplicativo, es posible operarlo mediante nuevas formas, en este caso un texto en lenguaje natural.

Por último se intentó construir un prototipo, utilizando LEA (Lightweight Eyetracking Algorithm) (LEA, 2011), una librería para el reconocimiento del movimiento ocular. Pero no se pudo concluir, debido a que la librería se encuentra en versión de desarrollo y posee serias inestabilidades al momento de realizar la detección; además esta librería solo retorna la dirección del ojo, lo cual limita el manejo de la aplicación a 9 opciones (las combinaciones de arriba, centro, abajo con izquierda, centro y derecha). Con éste intento fue notoria la necesidad de que los componentes a utilizar con UJI, deban retornar una cantidad considerable de información, para que la utilización del middleware sea factible, ya que como en el caso de LEA que solo devuelve una de nueve posibles opciones, es más fácil y mejor trabajar directamente con la librería, sin usar UJI.

7 CONCLUSIONES Y RECOMENDACIONES.

Al concluir esta investigación, se obtuvo una librería y/o un middleware para Java, cuya funcionalidad es hacer las veces de puente entre un motor de análisis de información no estructurada y una aplicación cliente de cualquier ámbito. Este puente encapsula la complejidad del funcionamiento del motor análisis (UIMA) y le brinda toda su funcionalidad de una manera sencilla a la aplicación cliente.

UJI interpreta el resultado del análisis generado por UIMA, convirtiendolo en una estructura de datos manejable, para luego compararla con el operationDocument y ejecutar cada uno de los métodos aquí contenidos.

Al simplificar la utilización de la plataforma UIMA, a la configuración de un archivo XML mediante UJI, se facilita de gran manera el trabajo del constructor de software, al momento de utilizar UIMA para implementar aplicativos manejados mediante información no estructurada, ya que sin ningún conocimiento adicional y sin un mayor esfuerzo por configurar la plataforma, es sencillo anexar a sus aplicativos la opción de operarlos mediante datos no estructurados.

Con los items anteriores se da solución a la pregunta: ¿Cómo facilitar la interpretación de los resultados del análisis generado en UIMA para que de esta manera constructores de software inexpertos en el área del procesamiento

de datos no estructurados, puedan añadir a sus aplicativos la opción de operarlos mediante información no estructurada?.

Con respecto a los objetivos planteados inicialmente en la investigación tenemos que:

1. Se propuso un modelo conceptual para diseñar y desarrollar aplicativos fácilmente integrables a la arquitectura UIMA, con el fin de utilizar el poder de análisis de información no estructurada que ésta arquitectura posee. En el numeral 5.5 se ilustró el funcionamiento de la arquitectura y micro-arquitectura planteada para la interfaz, así como cada componente que intervino en el diseño de este.

2. Se estudió el área del procesamiento de datos no estructurados y la representación de su análisis, investigando cómo acceder a esta última para transformarla en una estructura de datos manejable. En el capítulo 3.1.3 se documentó la representación del resultado del análisis en UIMA mediante CAS, para luego plantear en el numeral 5.5.4 en la micro-arquitectura el componente Result, que contiene una estructura de datos en forma de árbol para el manejo de la información.

3. Se encontró una forma para que solo fragmentos del resultado del análisis de la información que encajen con un determinado patrón, sean quienes desencadenen las acciones pertinentes en la aplicación.

Mediante la estructura evento-método planteada en el operationDocument, documentada en el capítulo 5.6 , se logró la posibilidad de crear patrones, constituidos por tipos y características asignados métodos y clases, logrando que cuando el resultado del análisis encaje con un evento se ejecute el respectivo método.

4. Se implementó un perfil operativo del modelo conceptual, que permitió validarlo y evaluó el acoplamiento de la interfaz a este. Este perfil fue denominado Opptalt y se documenta más abajo en éste mismo capítulo.

UJI le permite al constructor de software anexar a sus aplicativos la opción de operarlos mediante datos no estructurados, de una forma fácil y sin la necesidad de poseer ningún conocimiento en la configuración de UIMA o en el área del procesamiento de datos. UJI estimula y simplifica la utilización de UIMA en el área de la interacción persona-ordenador, ya que mediante UJI es posible crear aplicativos manejados mediante audio, video o textos en lenguaje natural, simplemente realizando una serie de pasos al alcance de cualquier desarrollador.

Gracias a la arquitectura utilizada en UJI, es posible reutilizar componentes de procesamiento de información desarrollados para UIMA, permitiendo beneficiarse de los adelantos de otras áreas y garantizando que la interfaz estará actualizada con los más recientes descubrimientos en el área del procesamiento de información no estructurada.

UJI beneficia a la industria del software, en el punto de que crea un puente de colaboración, que fluye desde el área del procesamiento de información hacia hacia este, es decir que los adelantos de última tecnología en aspectos de procesamientos de información no estructurada, irán creando nuevas y mejores interfaces de usuario, que anexarán un valor agregado a sus aplicativos

Para aprovechar las ventajas de ésta tesis, la academia podría orientar proyectos de investigación hacia el sector del procesamiento de información, específicamente a aspectos relacionados con procesos comunicativos humanos, tales como el lenguaje corporal y la comunicación verbal, y de esta forma crear componentes realmente útiles para una interacción persona-ordenador más optima

Como trabajo futuro se plantea la implementación de una herramienta para la construcción automática del operatiónDocument a partir de ontologías generadas por minería de datos, lo que garantizaría una mejor asignación de eventos a acciones y una más efectiva ejecución de los métodos en el momento indicado. Un ejemplo del uso de esta herramienta, puede ser aplicar minería de datos a conversaciones humanas, para luego generar ontologías y construir posteriormente el operationDocument; de esta forma se podría operar un aplicativo de una forma intuitiva y natural, permitiendo la flexibilidad sintáctica en las órdenes, propia del lenguaje natural.

A continuación se sugiere una serie de aplicativos que podrían implementarse fácilmente con UJI y los componentes de procesamiento adecuados:

- Software para personas con discapacidades físicas.
- Aplicativos para el control de maquinarias industriales.
- Software adaptatívos a estados anímicos.
- Aplicaciones para cajeros bancarios
- Domótica.
- Herramientas para la agilización de procesos ofimáticos.
- Robótica.

Para trabajos futuros también se recomienda la creación de un software asistente para la utilización de UJI, más concretamente un complemento para IDE's como Eclipse o Netbeans que facilitarían aun más el proceso. Con este tipo de complementos es posible crear y configurar los archivos necesarios de una forma fácil e intuitiva.

A pesar del hecho que la investigación se realizó en base a unos objetivos específicos, y que estos se cumplieron completamente; una posible limitación y una oportunidad de investigación, es el hecho de que UJI no fue probada en una población muestral de constructores de software, para de esta forma verificar la curva de aprendizaje y el porcentaje de adaptación a este.

En términos de resultados inesperados, el hecho de que componentes como LEA (LEA, 2011) resulten inestables al momento de usarlos, muestra que hay

que tener en consideración la calidad de los componentes antes de anexarlos a UJI.

Para concluir se plantea la necesidad de un gran repositorio de componentes reutilizables, en donde se centralicen los aportes alojados en dispersos servidores al rededor del mundo, brindando herramientas para el procesamiento de la información no estructurada y estimulando un ambiente propenso para el desarrollo de aplicativos multi-modales y la interacción persona-ordenador.

BIBLIOGRAFÍA

(Apache Licence, 2011) The Apache Software Foundation, , 2011, [ref. de Mayo de 2011], Disponible en Web: <http://www.apache.org/licenses/>

(Apache UIMA Development Community, 2009) Apache UIMA Development Community, UIMA Overview & SDK Setup., 2009

(ASF, 2011) The Apache Software Foundation, The Apache Software Foundation, , [ref. de Mayo 2011], Disponible en Web: <http://www.apache.org/>

(Baumgartner et al., 2008) William A Baumgartner Jr, K Bretonnel Cohen and Lawrence Hunter, An open-source framework for large-scale, flexible evaluation ofbiomedical text mining systems, 2008 Journal of Biomedical Discovery and Collaboration 2008, 3:1

(Bilhaut y Widlöcher, 2007) Bilhaut. F y A. Widlöcher, LinguaStream: An Integrated Environmentfor Computational Linguistics Experimentation, 2007 Universitaires de Grenoble

(BioNLP, 2010) , BioNLP UIMA Component Repository, , [ref. de Agosto de 2010], Disponible en Web: <http://bionlpuima.sourceforge.net/>

(Burgos Domingéz, 2002) Burgos Domingéz, A., Lingüística Computacional: Un Esbozo, 2002 Boletín de Lingüística

(Castelo et al., 2007) Castelo Daniel, Isi Jorge y Martínez Sebastián, WebQA Respuesta automática a preguntas, 2007

(Cruz et al., 2009) Fermín Cruz, J. a., Troyano, F. E., & Díaz, V. J., Construcción de un sistema PLN usando el framework UIMA, 2009 Procesamiento del Lenguaje Natural

(Cunningham et al., 2002) Cunningham. H, Maynard. D, Bontcheva. K ,

Tablan, V., GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications, 2002 Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics

(Diaper & Neville, 2004) Diaper, D., & Neville A, S., THE HANDBOOK OF TASK ANALYSIS FOR HUMAN-COMPUTER INTERACTION. New Jersey, 2004

(Eclipse, 2011) The Eclipse Foundation, , 2011, [ref. de Mayo de 2011], Disponible en Web: <http://www.eclipse.org/>

(Ellogon, 2010) Ellogon, Sitio Oficial, 2010, [ref. de Agosto de 2010], Disponible en Web: <www.ellogon.org/>

(Ferrucci, 2004) Ferrucci, D., Lally, A, UIMA. An architectural approach to unstructured information processing in the corporate research environment., 2004 In Natural Language Engineering

(Gantz, 2007) Gantz, J. F., A Forecast of Worldwide Information Growth Through 2010, 2007

(GATE, 2010) GATE, Sitio Oficial, 2010, [ref. de Agosto de 2010], Disponible en Web: <http://gate.ac.uk/>

(Gnjatović et al., 2008) Milan Gnjatović, Manuela Kunze y Dietmar Rösner, Processing dialogue-based data in the UIMA framework, 2008

(GoF, 1995) R Johnson, E Gamma, R Helm, J Vlissides, Design Patterns - Elements of Reusable Object-Oriented Software, 1995

(Gómez, 2000) Gómez Guinovart, J, Perspectivas de la lingüística Computacional., 2000 NOVATICA

(Goslin et al., 2000) Goslin, J. Joy, B. Steele, G. Bracha, G, The Java Language Specification, 2000

(GPL Licence, 2011) Free Software Foundation, GPL Licence, , [ref. de Mayo de 2011], Disponible en Web: <http://www.viti.es/gnu/licenses/gpl.html>

(Hahn et al. 2008)Udo Hahn, Thilo Götz, Eric W. Brown, Hamish Cunningham, Eric Nyberg, Towards Enhanced Interoperabilityfor Large HLT Systems:UIMA for NLP, 2008

(IBM, 2010) IBM, Uima, unstructured information management architecture, 2010, [ref. de Agosto de 2010], Disponible en Web: <http://www.research.ibm.com/UIMA/>

(Malone, 2007) Malone, R, Structuring Unstructured Data , 2007

(Mohamed Radhouene, 2008) Mohamed Radhouene Aniba, Sophie Siguenza, Anne Friedrich, Fre□de□ric Plewniak, Olivier Poch, Aron Marchler-Bauerand Julie DawnThompson, Knowledge-based expert systemsand a proof-of-concept case studyfor multiple sequence alignmentconstruction and analysis, 2008 BRIEFINGS IN BIOINFORMATICS.

(Netbeans Profiler, 2011) Netbeans, Netbeans Profiler, [ref. de Mayo 2011], Disponible en Web: <http://profiler.netbeans.org/>

(LinguaStream, 2010) LinguaStream, An integrated experimentation environment forcomputational linguistics., 2010, [ref. de Agosto de 2010], Disponible en Web: <http://www.linguastream.org/>

(LEA, 2011) lea-eyetracking, LEA - Lightweight Eyetracking Algorithm, [ref. de Mayo de 2011], Disponible en Web: <http://lea-eyetracking.sourceforge.net/>

(Lorés, 2002) Lorés, J, La interacción persona-ordenador

(Overview and Setup, 2009) The Apache UIMA Development Community, UIMA Overview & SDK Setup, 2009 The Apache Software Foundation

(Petasis et al., 2002) Petasis. G, Karkaletsis V, Paliouras. G, Androutsopoulos. I, Spyropoulos C., Ellogon: A New Text Engineering Platform, 2002 Third International Conference on Language Resources and Evaluation – LREC

(Poch, 2007) Poch, M. Cuestas, D. Mariño, J. Méndez, F. Sainz, I,

ARQUITECTURA DISTRIBUIDA PARA UN SISTEMA DE TRADUCCIÓN DELHABLASOBRE LA PLATAFORMA UIMA, 2008

(Rune Saetre et al., 2010) Rune Saetre, Kazuhiro Yoshida, Makoto Miwa, Takuya Matsuzaki, Yoshinobu Kano, and Jun'ichi Tsujii, Extracting Protein Interactions from Text withthe Unified AkaneRE Event Extraction System, 2010 IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, VOL. 7, NO. 3, JULY-SEPTEMBER 2010

(StarUML, 2011) StarUML, StarUML, , [ref. de Mayo de 2011], Disponible en Web: <http://staruml.sourceforge.net/en/>

(Techera, 2007): Techera, Cecilia , Lavinia: Ambiente Web para PLN, 2007

(UIMA Tools Guide and Reference) The Apache UIMA Development Community, UIMA Tools Guide and Reference, 2009 The Apache SoftwareFoundation

(W3C Multimodalidad, 2011) W3C, [ref. de Mayo de 2011], Disponible en Web: <http://www.w3c.es/divulgacion/guiasbreves/Multimodalidad>

(W3C, 2011) World Wide Web Consortium, World Wide Web Consortium, , [ref. de Mayo de 2011], Disponible en Web: <http://www.w3.org/>

(Wei-Bang, 2007) Wei-Bang Chen, Chengcui Zhang, Wen-Lin Liu, and Richa Tiwari, MIA: A UIMA-Based Microarray Image Analysis System, 2007 Ninth IEEE International Symposium on Multimedia 2007 - Workshops