Plataformas y sistemas de procesamiento lingüístico de alto rendimiento

Aitor Soroa
German Rigau
Jordi Porta
Jordi Atserias
Xavier Gómez Guinovart
Horacio Saggion

Resumen ejecutivo

Para catalizar el potencial de las tecnologías del lenguaje, el Ministerio de Industria, Energía y Turismo ha diseñado el Plan de Impulso de las Tecnologías del Lenguaje. Este Plan tiene por objeto fomentar el desarrollo del sector del Procesamiento del Lenguaje Natural (PLN) y la traducción automática en España. El Plan contempla entre sus líneas de actuación (línea 1 del Eje III) la creación de una plataforma de PLN en las Administraciones Públicas con el fin de incorporar estas tecnologías para mejorar la calidad y capacidad de los servicios públicos.

Como señala el Plan, se pretende "simplificar, lograr sinergias y aplicar la economía de escala en la puesta en marcha de nuevos servicios basados en tecnologías de procesamiento de lenguaje natural y traducción automática. Se busca el ahorro de costes mediante la compartición de recursos entre las Administraciones Públicas, y la eliminación de duplicidades y redundancias; además de mediante la mejora de la eficiencia de los procedimientos y la mejora del conocimiento. La plataforma se basará en el empleo de componentes reutilizables e interoperables, y, preferentemente, con licencias no restrictivas de código abierto". El Plan también señala que los primeros casos de uso serán los proyectos faro que también contempla el Plan (Eje IV), por lo que la sinergia entre ambas actuaciones es un requisito clave.

En este contexto, el objetivo de este informe es efectuar un estudio amplio sobre diferentes herramientas, técnicas, métodos y arquitecturas existentes para desarrollar e implantar sistemas de procesamiento lingüístico de alto rendimiento. La principal característica de estos sistemas es la de ser capaz de procesar lingüísticamente grandes volúmenes de texto dentro de unos límites temporales restringidos.

En el informe se analizan las principales tecnologías descritas en las áreas de procesadores de PLN, estándares de anotaciones, contenedores de anotaciones, cadenas de procesamiento, arquitecturas distribuidas, esquemas de ejecución, procesado paralelo, despliegue de sistemas, etc. El objetivo del estudio es doble. Por un lado, ofrecer un panorama actual del estado de estas tecnologías, y proporcionar una terminología común que ayude en la tarea de evaluar y comparar soluciones PLN dentro del Plan de Impulso de las Tecnologías del Lenguaje, perteneciente a la Agenda Digital para España. Por otro lado, el informe pretende ser un punto de partida para empresas que estén interesadas en el área de la minería de textos y PLN.

El informe ha sido realizado por una comisión de expertos de reconocido prestigio en el ámbito del PLN y sistemas distribuidos. Los autores del informe son los siguientes:

- **Aitor Soroa**, Universidad del País Vasco / Euskal Herriko Unibertsitatea
- **German Rigau**, Universidad del País Vasco / Euskal Herriko Unibertsitatea
- **Jordi Porta**, Centro de Estudios de la Real Academia Española
- **Jordi Atserias**, Trovit
- **Xavier Gómez Guinovart**, Universidade de Vigo
- **Horacio Saggion**, Universitat Pompeu Fabra

Índice general

1. Introducción

El presente informe analiza las principales tecnologías en el área del Procesamiento de Lenguaje Natural (PLN) para el procesamiento automático de grandes cantidades de texto en un tiempo limitado. El informe tiene dos objetivos generales. Por un lado, proporcionar una terminología común que ayuda en la tarea de evaluar y comparar soluciones PLN dentro del Plan de Impulso de las Tecnologías del Lenguaje, perteneciente a la Agenda Digital para España. Por otro lado, el informe ofrece un panorama del estado actual de las tecnologías, herramientas, técnicas y arquitecturas para desarrollar e implantar sistemas de procesamiento lingüístico de alto rendimiento. Así, tanto grupos innovadores en PLN, como empresas que estén interesadas en el área de la minería de textos y PLN, encontrarán en el presente informe un punto de partida para conocer la problemática específica del área, así como las técnicas y soluciones disponibles actualmente para obtener aplicaciones PLN que trabajen con gran cantidad de textos.

El área del PLN está en constante evolución y cada día surgen nuevos algoritmos, componentes y sistemas que ofrecen nuevas soluciones a la problemática general del análisis de textos. Así, el presente documento no pretende hacer recomendaciones que favorezcan ciertas soluciones sobre otras, ya que pensamos que estas recomendaciones pueden tener un alcance muy corto y quedar invalidadas en un breve espacio de tiempo. Entendemos, pues, que es más importante ofrecer al lector una serie de pautas y descripciones que le ayuden a seleccionar la mejor elección dentro de un abanico de posibilidades existentes.

Como veremos en el informe, las aplicaciones PLN complejas (extracción de información, análisis de sentimientos, búsqueda de respuestas, etc.) requieren integrar muchos módulos PLN que realizan el procesamiento lingüístico en varios niveles. Integrar estos módulos de forma efectiva es un área importante dentro de la ingeniería de la lengua y el PLN, y en este estudio analizaremos diversos factores que influyen en su correcta realización. El estudio es de carácter general, pero se centrará en los idiomas cooficiales del estado español, así como en el inglés.

Después de esta breve introducción, el informe comienza en el capítulo 2 con una

introducción general al PLN y presentando los principales niveles y tareas de procesamiento lingüístico. A continuación, el informe analiza diferentes alternativas actuales en los esquemas y formatos para la anotación textos. Los procesadores lingüísticos utilizan esquemas de anotación como una forma de contenedores de anotaciones con los que asocian información lingüística con partes del texto procesado o con anotaciones previas. En ocasiones estos contenedores de anotaciones permiten anotar formatos multimedia, versionar las anotaciones (permitiendo sobreponer anotaciones automáticas o la postedición por parte de operadores humanos), etc. El capítulo 3 describe las características de estos esquemas de anotación, así como una descripción de los formatos más utilizados actualmente.

En el capítulo 4 se estudian las principales herramientas y librerías PLN de carácter general, describiendo sus características, soporte de idiomas, entornos de ejecución, ciclo de vida, etc. En general, los sistemas descritos en este capítulo conforman *suites* PLN que se distribuyen con licencias libres, de tal manera que cualquiera puede descargarlas y ejecutarlas para realizar procesado básico PLN en varios idiomas.

El capítulo 5 analiza los sistemas actuales para describir y ejecutar flujos de trabajo, o *workflows*, con el objetivo de implementar tareas PLN complejas. Como hemos indicado anteriormente, las aplicaciones PLN están formadas por módulos lingüísticos básicos, integrados formando un flujo de trabajo concreto. Este desarrollo modular, orientado a componentes interoperables, hace que la especificación de las cadenas de procesamiento sea una tarea imprescindible para el procesamiento textual de distintas tipologías de documentos. En el capítulo se analizan los entornos de procesamiento más relevantes, que permiten definir flujos de trabajo de forma declarativa, y ofrecen un ejecutarlos dentro de un marco común.

Por otro lado, la cantidad de información en forma de texto accesible actualmente es enorme, y no para de crecer. El procesamiento de toda esa cantidad de información textual requiere incrementar la capacidad de potencia de cálculo y adoptar nuevos paradigmas de procesamiento que permitan el procesado lingüístico escalable a gran escala. Así, los flujos de trabajo o *workflows* PLN pueden ser procesados siguiendo distintos esquemas de ejecución, como son la utilización de colas, sistemas de procesamiento distribuido de tuplas, sistemas de computación paralela con datos locales, etc. El capítulo 6 describe estos esquemas de computación de aplicaciones, incluyendo métodos para desplegar sistemas PLN en *clusters* de ordenadores.

Entre las nuevas aportaciones de los métodos modernos de computación se encuentran las grandes plataformas de computación en la nube. Éstas emplean componentes escalables para la construcción de aplicaciones escalables. El capítulo 7 estudia las principales plataformas en el área PLN, y la facilidad de creación de componentes compatibles con las mismas así como la puesta a disposición de grandes colecciones documentales.

2. Sistemas de procesamiento PLN

Los sistemas informáticos procesan muy fácilmente datos estructurados. Es decir, información que tiene una estructura y un significado único, explícito y completo. Así, los sistemas informáticos actuales pueden manejar fácilmente bases de datos con millones de registros y datos numéricos. Sin embargo, como el lenguaje natural es la forma más común y versátil que tiene la humanidad de transmitir información, más de 90 % de la información digital disponible es información no estructurada en forma de textos y documentos (escritos o hablados) en múltiples lenguas[1]. Para los sistemas informáticos el problema radica en que toda esta información textual está sujeta a múltiples interpretaciones ya que suele ser incompleta y ambigua. Para entender la información que contienen estos textos, una capacidad reservada hasta ahora a los humanos, es necesario seleccionar la interpretación correcta y hacer explícita su relación tácita con la realidad.

El desarrollo de sistemas capaces de procesar y entender las expresiones textuales que aparecen en los textos de cualquier lengua y dominio ha sido un reto para la lingüística e informática desde hace muchos años. Al ser un problema muy complejo y difícil, éste ha sido subdividido en tareas o niveles de procesamiento menos complejas que pueden ser abordadas por módulos especializados. Normalmente, tareas de anotación más complejas (por ejemplo, el análisis sintáctico de las oraciones) pueden basarse en los resultados obtenidos anteriormente por otras más sencillas (por ejemplo, segmentación en frases y palabras, etiquetado morfosintáctico). Evidentemente, esta interdependencia entre las tareas implica también una interdependencia en los resultados.

Así, uno de los principales problemas a los que deben enfrentar los desarrolladores e investigadores en este área es que para su comprensión completa, los textos deben ser analizados con precisión en todos los niveles necesarios que requiera la aplicación final. Por otra parte, cada uno de estos niveles se ven afectados por expresiones ambiguas que

[1] http://www.coveo.com/~/media/Files/WhitePapers/Coveo_IDC_Knowledge_Quotient_June2014.ashx

no pueden ser interpretados de manera aislada. Abordar la ambigüedad y la imprecisión inherente al lenguaje natural es el objetivo principal de varias tareas de PLN. Por otra parte, todas estas tareas de PLN se tienen que abordar actualmente idioma a idioma, y dominio a dominio.

2.1 Niveles de PLN

Los niveles principales de procesamiento del lenguaje natural textual se corresponden con los principales del estudio del lenguaje: fonológico, morfológico, sintáctico, semántico y pragmático.

- *Fonológico*: este nivel estudia los sonidos de una lengua.
- *Morfológico*: es la rama de la lingüística que estudia la estructura interna de las palabras para delimitar, definir y clasificar sus unidades, las clases de palabras a las que da lugar (morfología flexiva) y la formación de nuevas palabras (morfología léxica).
- *Sintaxis*: estudia las reglas que gobiernan la combinatoria de constituyentes sintácticos y oraciones. La sintaxis, por tanto, estudia las formas en que se combinan las palabras para obtener estructuras comunicativas.
- *Semántico*: estudia los aspectos del significado, sentido o interpretación de los símbolos, palabras, expresiones, frases o estructuras discursivas más complejas.
- *Pragmático*: estudia el modo en el que el contexto influye en la interpretación del significado. Evidentemente, el lenguaje forma parte de un sistema de interacción social con sus propias convenciones y reglas que tienen sentido en una cierta comunidad cultural.

Así, el procesamiento de textos puede requerir realizar tareas básicas a nivel de documento tales como la identificación del idioma (por ejemplo, castellano, inglés, etc.), su clasificación (por ejemplo, detección de correo spam), la segmentación del documento en frases, párrafos, capítulos, detección de estructuras discursivas, etc., tokenización o selección de los elementos (o tokens) que componen un texto, palabras o multipalabras del texto, etiquetado de la categoría morfosintáctica (por ejemplo, dependiendo del contexto, la forma *partidos* puede ser un sustantivo plural o un participio masculino plural), lematización o búsqueda de las formas canónicas de las palabras (por ejemplo, la forma *partidos* puede venir del verbo *partir* o del sustantivo *partido*), desambiguación del sentido de las palabras (por ejemplo, un *partido* puede ser un evento deportivo o una agrupación política), análisis sintáctico para detectar los componentes sintácticos de la oración, etc. así como tareas más avanzadas a nivel de oración tales como la detección y clasificación de las entidades nombradas (nombres própios), y su vinculación a un repositorio de entidades como Wikipedia, detección e interpretación unívoca de expressiones temporales, numéricas, etc., identificación de los predicados y participantes junto con sus roles semánticos, o más allá del ámbito de la oración tareas como la correferencia nominal, eventiva, la identificación de roles implícitos, estructuras discursivas, etc.

A modo de ejemplo, el entregable *D4.1 Resources and linguistic processors*[2] del proyecto europeo NewsReader [3] [Vos+16] compila un buen número de tareas de PLN y algunos de los procesadores más avanzados que las abordan para los cuatro idiomas

[2] `http://kyoto.let.vu.nl/newsreader_deliverables/NWR-D4-1.pdf`
[3] `http://www.newsreader-project.eu`

contemplados en el proyecto y que al integrarlas han dado lugar a las cadenas de procesamiento con decenas de modulos descritas en el entregable *D4.2.3 Event detection, v3*[4]. Estas cadenas de procesamiento realizan múltiples tareas de anotación a distintos niveles. Los niveles de anotación más básicos permiten realizar anotaciones lingüísticas de más alto nivel. Como ya hemos mencionado anteriormente, este encadenamiento de componentes produce una propagación de errores. Por tanto, los errores en niveles básicos de procesamiento (por ejemplo, tokenización, lematización, etiquetado morfosintáctico, etc.) tienen un efecto directo en los errores de niveles lingüisticos de más alto nivel.

Evidentemente, el conjunto de nuevas tareas crece continuamente según surgen nuevas necesidades, intereses, y tecnología para abordarlas. Así la lista de componentes y sistemas de PLN tampoco deja de crecer. También es necesario aclarar que estas tareas son genéricas y que cada dominio de especialización (por ejemplo, médico, jurídico, etc.) puede requerir tareas específicas.

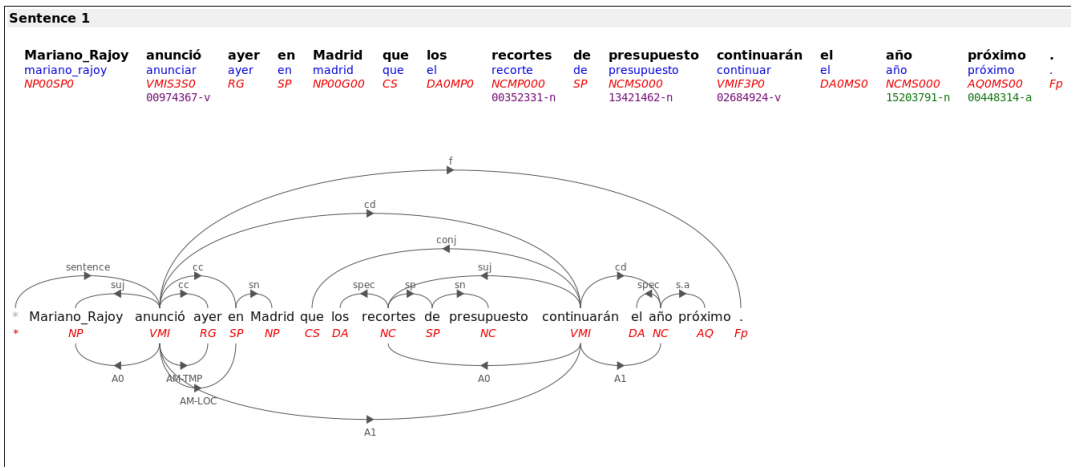

Figura 2.1: Anotaciones lingüísticas que proporciona para el castellano el software Freeling disponible en `http://nlp.lsi.upc.edu/freeling/demo/demo.php`.

También a modo de ejemplo, en la figura 2.1 mostramos el grafo que obtiene Freeling [PS12] tras aplicar entre otros los siguientes módulos: un detector de idioma, un tokenizador, un etiquetador morfosintáctico, un detector y clasificador de entidades nombradas, un desambiguador del sentido de las palabras, una analizador sintáctico y un etiquetador de roles semánticos.

2.2 Tareas de PLN

En esta sección describiremos con un poco más de detalle algunas de las tareas de PLN más usuales. De todos modos, como ya hemos indicado, estas tareas no deben verse de forma aislada, sino como tareas interdependientes ya que las anotaciones realizadas en una tarea, luego son reutilizadas en tareas posteriores.

Tokenización y segmentación

La tokenización (en inglés *Tokenization*) consiste en la segmentación del texto original en *tokens*, es decir, las palabras, números, símbolos, etc. que aparecen en el texto. El

[4] `http://kyoto.let.vu.nl/newsreader_deliverables/NWR-D4-2-3.pdf`

tokenizador es así el encargado de identificar estas palabras y calcular sus correspondientes posiciones en el texto, en forma de *offset*s. Además, el tokenizador se encarga a menudo de reconocer las frases y párrafos del texto.

Análisis morfosintáctico. Lematización y categoría gramatical.

El análisis morfosintáctico (en inglés *Part-of-Speech tagging*) consiste en asignar información de carácter morfológico a los tokens. En el caso más básico, éstos analizadores identifican el lema del token (la raíz de la palabra o forma canónica) y le asignan una categoría sintáctica. Además, muchas veces se asigna también información morfosintáctica más compleja tal como el género, número, tiempo verbal, declinación, etc. Por último, esta tarea suele ser también la responsable de identificar unidades multi-palabra, conjunto de tokens que funcionan como una sola unidad léxica (por ejemplo, "hombre rana", "tomar el pelo", etc.).

Anotaciones sintácticas

El análisis sintáctico (en inglés *Syntactic Parsing*) se encarga de realizar las anotaciones sintácticas que incluyen los árboles sintácticos de sintagmas y frases. Existen multitud de formulaciones y teorías sintácticas, algunas más profundas y otras más superficiales. Usualmente se diferencian tres tipos de anotaciones sintácticas. El *chunking* es la identificación superficial de sintagmas, frases subordinadas, etc. El *análisis de dependencias* es un formalismo sintáctico donde los árboles están formados por medio de relaciones binarias entre palabras. Por último, el *análisis de constituyentes* representa la oración como un árbol de derivación sintáctica.

Asignación de sentidos

Dentro de las anotaciones semánticas, la anotación de sentidos es el resultado de aplicar la llamada desambiguación de acepciones (en inglés *Word Sense Disambiguation*). Muchas palabras del lenguaje son ambiguas (por ejemplo, la palabra "vela" puede ser, entre otros, un cilindro de cera usado para alumbrar o una lona que se utiliza en los barcos para impulsarlos). La desambiguación de acepciones consiste en asignar a cada palabra su acepción correcta dependiendo del contexto en el que ésta aparezca. Normalmente se usan repositorios de sentidos como diccionarios o más comunmente *WordNet*[5] [Fel98] o para las lenguas peninsulares el *Multilingual Central Repository*[6] [GLR12].

Identificación, clasificación y desambiguación de entidades nombradas

Las entidades nombradas son expresiones que se refieren a nombres propios, incluyendo personas, organizaciones y lugares, y expresiones numéricas tales como fechas, cantidades, dinero, etc. Los sistemas de reconocimiento y clasificación de entidades nombradas (del inglés NERC o *Named Entity Recognition and Classification*) reconocen expresiones sobre entidades nombradas en el texto y las clasifican según su categoría (persona, lugar, organización, etc.). Adicionalmente, los sistemas de desambiguación de entidades nombradas (*Named Entity disambiguation*) o NEL (*Named Entity Linking*) asocian la entidad nombrada con un elemento de una base de datos externa como Wikipedia, Dbpedia, Wikidata, etc.

[5] `http://wordnet.princeton.edu`

[6] `http://adimen.si.ehu.es/web/MCR`

Análisis de sentimientos

El análisis de sentimientos (del inglés *Sentiment Analysis*) y la minería de opiniones (del inglés *Opinion Mining*) se ocupa de analizar las opiniones, sentimientos, valoraciones, actitudes y emociones en el texto [Liu12]. Es una tarea útil para las organizaciones que quieren saber cómo su marca o producto es percibido por el público, por ejemplo, en las redes sociales, blogs o en evaluaciones y comentarios de clientes.

Asignación de roles semánticos

La tarea llamada *Semantic Role Labeling* (SRL) consiste en detectar los argumentos semánticos asociados con los verbos o elementos predicativos en el texto, y clasificarlos en sus respectivos roles. Por ejemplo, en la oración "Manuela vendió el libro a Juana", un sistema SRL reconocería que "vender" es el predicado principal, que "Manuela" cumple el rol de agente (o vendedora), que "libro" es el tema (u objeto de la transacción) y que "Juana" es el receptor (o compradora). Los roles semánticos representan un nivel de abstracción superior a los árboles sintácticos, ya que oraciones con construcciones sintácticas variadas pueden producir la misma asignación de roles semánticos. Por ejemplo, la frase "El libro fue vendido a Juana por Manuela" tiene una estructura sintáctica diferente pero los mismos roles semánticos. La asignación de roles semánticos suele ser un proceso indispensable en los sistemas de extracción de información, ya que éstos se corresponden con los elementos predicativos hallados en el texto.

En las anotaciones, los roles semánticos suelen requerir varios aspectos:

- Identificar las palabras tanto del predicado como de los argumentos.
- Asociar el predicado principal con un elemento de una base de datos externa (por ejemplo, PropBank [PGK05] o FrameNet [BFL97]).
- Clasificar cada argumento con el rol correspondiente.

Correferencia nominal y eventiva

La correferencia (en inglés *coreference*) es la identificación de menciones en el texto que comparten un mismo referente en el mundo real. Por ejemplo, las menciones "Barack Obama", "presidente Obama", "Mr. Obama", etc. se refieren todas al ex-presidente de EEUU Barack Obama. Existen multitud de variantes en el tratamiento de la correferencia (un caso particular del fenómeno lingüístico llamado anáfora), entre las que cabe destacar la correferencia nominal, que es la más común. También puede ser relevante tratar la correferencia eventiva cuando un mismo evento es mencionado varias veces en forma verbal o nominal en uno o múltiples documentos.

Detección e interpretación de expresiones temporales

Las expresiones temporales son secuencias de palabras que indican cuándo sucedió algo, su duración o el número de veces que ha ocurrido. Por ejemplo, .ªyer.º .el año próximo", en la figura 2.1 son expresiones temporales que pueden normalizarse a una fecha y año concretos si la fecha del enunciado es conocida. Éste es un tipo de anotación indispensable en sistemas que pretendan reconstruir la secuencia temporal de los eventos que se relatan en los documentos.

Detección de estructuras discursivas

Los objetos de análisis del discurso (discurso, escritura, conversación, evento comunicativo) se definen de diversas maneras en términos de secuencias coherentes de frases o

diálogos.

2.3 Interoperabilidad

Las aplicaciones de PLN se construyen sobre tareas más simples y relativamente bien definidas. Por ejemplo, la extracción de relaciones entre entidades nombradas en un texto viene a menudo precedida por la segmentación del texto, su etiquetado morfosintáctico, el reconocimiento de entidades y un análisis sintáctico de dependencias. Para que los distintos módulos de PLN interaccionen correctamente entre ellos y se obtenga el resultado esperado por su aplicación coordinada, es necesario que el formato y contenido de cada módulo sea el que esperan los módulos subsiguientes. Uno de los mayores problemas con estas herramientas es que, generalmente, no son compatibles entre ellas. Como veremos, la interoperabilidad de componentes representa un problema transversal en toda aplicación PLN que además se presenta en muchos niveles. Por ejemplo, la incompatibilidad de módulos se puede producir no ya por que utilicen formatos diferentes para representar las anotaciones, sino por el significado que cada componente atribuye a las propias anotaciones. Por poner un ejemplo, puede que un analizador sintáctico haya sido entrenado en un corpus sin multipalabras anotadas, pero puede que la anotación morfológica sí las tenga presentes y las haya anotado.

2.4 Recuperación y procesado de documentos

Existen también un conjunto de técnicas y procesos de recuperación y manipulación informática de documentos que históricamente han pertenecido al ámbito de la recuperación de la información (en inglés, *Information Retrieval*). Básicamente, la búsqueda de documentos por palabras clave dentro de una base de datos documental o en Internet, aunque también la clasificación o direccionamiento de textos, la agrupación de textos similares (en inglés *clustering*), etc. Aunque estas técnicas suelen trabajar con todo el documento (o párrafos o secciones completas) también requieren de un preprocesado básico de los textos. Normalmente, tokenización, lematización o técnicas básicas de normalización de la variedad morfológica como *stemming*.

Clasificación automática de textos

La clasificación automática de textos (en inglés *Text Classification*) implica la asignación de forma automática de un documento a un conjunto de clases predefinidas [DZ11]. La clasificación automática de textos tiene importantes aplicaciones en la gestión y clasificación temática de contenidos, direccionamiento de documentos, filtrado de correos basura, minería de opiniones, etc.

Agrupación automática de textos

La agrupación automática de documentos (en inglés *document clustering*) es la aplicación a documentos textuales de técnicas de agrupamiento (*clustering*). Tiene aplicaciones en la organización automática de documentos, extracción de tópicos y recuperación rápida de información o filtrado.

3. Representaciones Lingüísticas

Los procesadores PLN enriquecen los documentos textuales por medio de las llamadas anotaciones lingüísticas, que asocian información lingüística (lemas, categorías morfosintácticas, árboles sintácticos, etc.) a unidades de texto del documento original. La representación de anotaciones lingüísticas es un área compleja dentro del PLN que hereda los problemas propios de la información lingüística que se anota. Hoy en día existen muchas formas distintas de representar las anotaciones lingüísticas, que varían entre formatos propios utilizados por procesadores específicos hasta intentos de definir esquemas de carácter general. En este capítulo veremos los principales formatos de representación existentes para representar las anotaciones lingüísticas.

Las aplicaciones PLN necesitan a menudo integrar procesadores lingüísticos básicos (tokenización, etiquetado de categorías, reconocimiento de entidades nombradas, etc.) que realizan el análisis lingüístico a diferentes niveles. Aplicaciones complejas tales como la extracción de eventos, sistemas de pregunta-respuesta, etc., están construidas por medio de estas unidades básicas de proceso, que deben ser combinadas formando flujos de trabajo complejos. No obstante, los procesadores PLN están a menudo diseñados para resolver problemas específicos y son difíciles de integrar. Como se ha visto en la sección 2.3, una de las dificultades principales a la hora de combinar módulos PLN es la falta de interoperabilidad, causada en gran medida porque cada módulo utiliza sus propios esquemas y formatos para representar sus anotaciones. La interoperabilidad de las anotaciones es, pues, un aspecto fundamental para la compartición y reutilización de información en sistemas de procesamiento con distintos componentes. Es importante resaltar que la interoperabilidad se consigue con la adopción no sólo de los formatos de anotación, sino de las especificaciones y modelos lingüísticos comunes.

En el área del PLN han habido numerosos intentos de establecer estándares de representación lingüística con el objetivo de facilitar la reusabilidad e integración de recursos y procesadores lingüísticos [IR04]. Los estándares de anotación acostumbran a acompañarse con guías de anotación, documentos en los que se describen los criterios empleados

para anotar correctamente diferentes fenómenos lingüísticos. La adopción de modelos o especificaciones preexistentes tiene como ventaja sobre la elaboración de nuevas especificaciones la eliminación de la etapa de discusión y debate, a veces durante años, sobre los aspectos del modelado y representación de la información, y la reutilización de recursos anotados con dichas especificaciones. La ventaja de adoptar especificaciones de anotación fundamentadas en teorías lingüísticas consolidadas, fruto de años de investigación, facilita la labor de comprensión y distribución de las anotaciones.

La falta histórica de estándares *de jure*, desarrollados por un comité para alguna organización de estandarización[1] y que cubra todos los aspectos de una anotación, ha motivado el desarrollo, por parte de comunidades de desarrolladores, de estándares *de facto* para determinadas tareas. Algunos de estos estándares acaban siendo difundidos a través de la literatura científica y adoptados por otros. En general, cualquiera que sea el estándar, la adopción de un solo esquema plantea, entre otros, los siguientes problemas:

- Los objetivos de un estándar de anotación lingüística son muchos, y muchas veces contradictorios. Por un lado, un estándar de anotación debe ser capaz de representar una gran cantidad de fenómenos lingüísticos, pero a la vez debe hacerlo de una forma rígida y no ambigua, de tal manera que los usuarios sepan cómo trasladar las representaciones internas de los módulos a la representación estándar.
- El PLN es un área activa de investigación y cada día aparecen nuevas técnicas y aplicaciones. Es muy difícil, si no imposible, concebir una representación uniforme que pueda abarcar todos los niveles lingüísticos requeridos hoy en día.
- Muchos esquemas de representación carecen de un conjunto de librerías y APIs que permitan construir aplicaciones de forma efectiva.
- Muchos esquemas de anotación se ajustan a una cierta lengua o familia de lenguas, lo cual dificulta la adopción de estos esquemas en lenguas con diferentes características morfosintácticas.

En el resto del capítulo hablaremos, pues, de los formatos de anotación lingüística. Comenzaremos en la sección 3.1 definiendo conceptos básicos y la terminología que utilizaremos, como por ejemplo la distinción entre modelo y esquema de anotación. En la sección 3.2 describiendo cuestiones relevantes que hay que considerar a la hora de comparar formatos de anotación y, seguidamente, hablaremos de estándares para representar la información lingüística, es decir, la disponibilidad de etiquetas estándares para la representación de los análisis lingüísticos en varios niveles (sección 3.3). Después describiremos los principales formatos de anotación existentes. En la sección 3.4 y describiremos los modelos y esquemas de anotación más utilizados actualmente, incluyendo los estándares ISO más relevantes. En la sección 3.6 hablaremos de formatos de anotación específicos, formatos *ad-hoc* definidos para solucionar problemas concretos, y que no tienen usualmente vocación de estándar. Finalizaremos el capítulo con una tabla comparativa con una comparación de los diferentes formatos.

[1] Por ejemplo La Organización Internacional de Estándares (ISO), que es la responsable de la creación de estándares y normas usadas a nivel internacional para asegurar la compatibilidad e interoperabilidad entre la industria y las administraciones públicas, tanto a nivel nacional como internacional.

3.1 Anotaciones: modelos, esquemas e interoperabilidad

El término de formatos de anotación engloba diversos conceptos que, aunque fuertemente relacionados entre sí, describen básicamente aspectos diferentes. En esta sección aclararemos la terminología que utilizaremos durante el capítulo, resaltando las diferencias entre estos conceptos.

El *modelo de anotación* es la definición de una arquitectura común y un modelo de datos para representar anotaciones lingüísticas, que sea independiente de la aplicaciones PLN. El modelo de anotación define la forma general de las anotaciones, su organización jerárquica de distintos niveles, la forma de representar la ambigüedad de los análisis, etc. Ejemplos de modelos de anotación (también llamados contenedores de anotaciones) son *LAF* (ver sección 3.4.3), *TIPSTER* (ver sección 3.4.1) o *UIMA CAS* (sección 3.4.2). Una particularidad del modelo de anotación es que no propone qué elementos utilizar anotar fenómenos lingüísticos determinados. El modelo es simplemente el marco común para representar cualquier anotación.

El *esquema de anotación*, va más allá y define, junto al modelo general, la manera concreta de representar las anotaciones en diferentes niveles. De esta manera, los esquemas de anotación describen el conjunto de objetos y etiquetas a utilizar para representar anotaciones lingüísticas de diferentes niveles, es decir, anotaciones de tokens, análisis morfosintáctico, semántico, etc. Ejemplos de formatos de anotación son *NAF* (sección 3.5.1), *FoLia* (sección 3.5.2) o *NXT* (sección 3.4.4).

La adopción de modelos o esquemas de anotación ofrecen una solución a la llamada *interoperabilidad estructural*, que permite que diferentes módulos PLN puedan compartir las anotaciones utilizando un modelo de datos común. Por desgracia, la interoperabilidad estructural por sí sola no es suficiente para conseguir una integración real de componentes PLN. En efecto, el contenido de propias anotaciones y su significado concreto debe ser también compartido por las distintas aplicaciones, la llamada *interoperabilidad conceptual*. La interoperabilidad conceptual está directamente relacionada con el modelo lingüístico de análisis y su representación. Un analizador morfosintáctico puede, por ejemplo, anotar la palabra “llegó” como una forma de la tercera persona del pretérito indefinido del verbo “llegar”, y los subsiguientes módulos PLN deben conocer el significado de estas anotaciones.

3.2 Cuestiones sobre las representaciones lingüísticas

Existen diferentes factores a considerar a la hora de hacer comparaciones entre formatos de anotación. A menudo estos factores son una muestra de las diferentes soluciones adoptadas por la comunidad para hacer frente a los problemas que surgen para anotar lingüísticamente textos en formato electrónico. En esta sección vamos a ver de forma breve algunos de estos factores, que muchas veces son la causa de que los formatos de anotación tengan unas características u otras. La sección no pretende ser exhaustiva, ni nos proponemos enumerar toda la problemática que se da a la hora de anotar textos lingüísticamente. Adoptando una posición ciertamente pragmática, describiremos sólo aquellas características que nos parecen más pertinentes dentro del ámbito del presente informe.

Anotación en línea versus *stand-off*

```
<TimeML>
<TEXT><TIMEX3 tid="t196" type="DATE" value="PRESENT_REF">Now</TIMEX3> with
new construction under way, three of his buyers have backed out. And Wong
Kwan will be lucky to break even.  Pak can't find buyers. She estimates her
properties, worth a hundred thirty million dollars in <TIMEX3 tid="t89"
type="DATE" value="1997-10">October</TIMEX3>, are worth only half that
<TIMEX3 tid="t90" type="DATE" value="PRESENT_REF" >now</TIMEX3>.</TEXT>
</TimeML>
```

Figura 3.1: Ejemplo de anotación en línea

Como hemos dicho anteriormente, las anotaciones lingüísticas asocian información lingüística a unidades de texto del documento original. Existen dos alternativas básicas para representar estas anotaciones: la anotación en línea (*inline*) y la llamada anotación *stand-off*.

En la anotación en línea las anotaciones se insertan físicamente dentro del documento y cada palabra o porción de texto de interés es anotado con la información correspondiente. La figura 3.1 muestra un ejemplo del esquema de anotación *TimeML*, que adopta este paradigma. Como se puede ver, las marcas de anotación están insertadas en el texto y cubren la porción de texto que se quiere anotar (en este caso, expresiones temporales), añadiendo información temporal al mismo.

En el tipo de anotación *stand-off* el documento original no se modifica, y las anotaciones se construyen asociándolas al texto original por medio de *anclas* (*anchors*) que hacen referencia al documento original. A menudo, esta referencia se construye describiendo la posición en el documento de la porción de texto a anotar, los llamados *offsets*.

La figura 3.2 muestra un ejemplo de anotación *stand-off*. Se puede ver el contenido del documento original (elemento `raw`), y cómo los componentes léxicos son descritos por medio de *offsets*. Análisis subsecuentes se realizan utilizando identificadores unívocos a anotaciones anteriores. Por ejemplo, la anotación de lemas y categorías gramaticales se anclan a los componentes léxicos anotados previamente por medio de los identificadores de éstos últimos.

La anotación en línea, al contrario de la anotación *stand-off*, no necesita mantener las localizaciones de las etiquetas sobre el texto, puesto que estas anotaciones están insertas en el propio texto afectado, rodeándolo. Por otro lado, la anotación *stand-off* es más flexible que la anotación en línea [Ana+01] y la mayoría de modelos de anotación de carácter general se adhieren a este principio. Las principales ventajas del esquema *stand-off* son las siguientes:

- El documento original queda intacto y se puede reconstruir de forma íntegra.
- Permite la representación de anotaciones sobre porciones de texto que se solapan. Además, ofrecen una alternativa más elegante para la representación de anotaciones ambiguas, es decir, aquellas que asocian más de una posibilidad a una misma porción del texto.
- Se pueden formar anotaciones jerárquicas y, así, permitir que niveles superiores de análisis hagan referencia a anotaciones creadas en niveles inferiores.

Codificación de caracteres

La correcta codificación de caracteres en los documentos textuales es un problema real que afecta a muchas plataformas de procesamiento PLN. Usualmente considerado un problema menor, la correcta gestión de la codificación de caracteres es muchas veces

```
<NAF xml:lang="en" version="v3">
<raw><![CDATA[Followers of Muqtada al-Sadr clashed with British troops in
  the city of Amarah in battles late Monday that killed 15 Iraqis and
  wounded eight, said a coalition spokesman in the city, Wun Hornbyckle.]]>
  </raw>
  <text>
    <wf id="w1" sent="1" offset="0" length="9">Followers</wf>
    <wf id="w2" sent="1" offset="10" length="2">of</wf>
    <wf id="w3" sent="1" offset="13" length="7">Muqtada</wf>
    ...
  </text>
  <terms>
    <!--Followers-->
    <term id="t1" type="open" lemma="follower" pos="N" morphofeat="NNS">
      <span><target id="w1"/></span>
    </term>
    <!--of-->
    <term id="t2" type="close" lemma="of" pos="P" morphofeat="IN">
      <span><target id="w2"/></span>
    </term>
    ...
  </terms>
</NAF>
```

Figura 3.2: Ejemplo de anotación stand-off

uno de los problemas más visibles a la hora de combinar diferentes módulos PLN en aplicaciones complejas. Este problema es particularmente común cuando se procesan textos en lenguas diferentes del inglés ya que, a menudo, los sistemas estan preparados para procesar textos en este idioma y fallan al tratar textos de lenguas con elementos tipográficos complejos, tales como tildes, diéresis, etc.

Las consecuencias de los problemas derivados de una incorrecta gestión de la codificación de caracteres son diversas, y van desde una visualización defectuosa de ciertos caracteres al cálculo incorrecto de las posiciones de las palabras (*offsets*). En efecto, a menudo los *offsets* representan la distancia desde el comienzo del documento medida en caracteres, pero ciertos caracteres se codifican utilizando secuencias de bytes (por ejemplo, el carácter 'ñ' en *UTF8* se codifica en la secuencia de dos bytes `#xC3 #xB1`). Así pues, es imprescindible conocer el esquema de codificación para poder transformar los *offsets* representados en caracteres a bytes reales dentro del documento.

Una de las causas principales de los problemas de codificación es que los documentos no suelen describir de forma explícita el esquema de codificación utilizado para representar los caracteres. Como veremos, este problema se ve aliviado utilizando formatos de serialización adecuados.

Formato de serialización

La *serialización* de las anotaciones lingüísticas es el proceso de convertir la representación interna (en forma de estructuras de datos que residen en memoria) a un formato que pueda ser compartido entre diferentes procesadores lingüísticos, que además pueden residir en diferentes nodos físicos. A menudo, la serialización se efectúa utilizando un formato estándar de intercambio entre procesadores, como por ejemplo XML. Ciertos esquemas de anotación permiten también serializar las anotaciones utilizando un formato propio *ad-hoc*.

Sin duda, el lenguaje de marcas XML es el formato de intercambio más utilizado actualmente para serializar anotaciones lingüísticas. Las ventajas de utilizar XML como

formato de serialización son las siguientes:

- Los formatos pueden utilizarse para representar anotaciones siguiendo diferentes paradigmas (anotación en línea o anotación *stand-off*).
- Están específicamente diseñados para ser compartidos entre diferentes plataformas/arquitecturas.
- Los desarrolladores de módulos PLN pueden utilizar librerías de carácter general para leer y escribir las anotaciones[2].
- Los formatos describen de forma explícita la codificación particular utilizada para representar los caracteres.

Si bien hoy en día la gran mayoría de sistemas utilizan XML como formato de serialización, el formato JSON está adquiriendo últimamente una cierta relevancia en el área. La principal ventaja de JSON radica en su estructura rígida y sencillez de análisis, lo cual permite reducir considerablemente el tiempo empleado en serializar/de-serializar las anotaciones. Por otro lado, muchas bases de datos de tipo NoSQL admiten documentos en formato JSON de forma nativa, de tal manera que grandes cantidades de anotaciones JSON pueden ser almacenadas en este tipo de bases de datos de forma eficiente. Así, muchos sistemas de procesamiento lingüístico de alto rendimiento adoptan JSON como formato de intercambio de anotaciones entre módulos. Especial atención merece el formato JSON-LD[3], una recomendación *World Wide Web Consortium* (W3C) que utiliza la sintaxis JSON para representar documentos y objetos enlazados *Linked Data*.

Aún así, JSON presenta una serie de carencias con respecto a XML, entre las que cabe destacar la dificultad para representar atributos de forma directa en las relaciones. Por otro lado, JSON esta orientado para ser utilizado entre ordenadores, y a menudo los documentos JSON son difíciles de interpretar para el observador humano. La imposibilidad de insertar comentarios dentro de los documentos JSON es un ejemplo de ello, ya que los comentarios cumplen la función de ayudar a las personas a entender el contenido de las anotaciones.

Por último, ciertos esquemas de anotación (p.ej., *NIF*, ver sección 3.4.5) utilizan el llamado *Resource Description Framework* (RDF, [LS99]) para representar las anotaciones lingüísticas. RDF fue desarrollado inicialmente para proveer vías formales para describir recursos (p.ej, personas físicas, libros en una biblioteca, etc.), pero su uso se ha extendido a numerosos escenarios. RDF está basado en la noción de tripletas, que están formadas por un predicado que enlaza un sujeto con un objeto, y que forman un grafo dirigido con arcos etiquetados. Todos los elementos de las tripletas se representan utilizando los llamados *URIs* (*Uniform Resource identifiers*) , identificadores globales que representan un *recurso* (persona, libro, etc.) de forma unívoca (de forma alternativa, los objetos de las tripletas pueden pertenecer a tipos básicos como números o cadenas de caracteres). RDF es la estructura de datos básica de la llamada *Semantic Web* y dispone de una comunidad amplia que lo mantiene. Debido a sus características, RDF proporciona ventajas para la publicación de recursos lingüísticos, tales como la representación basada en grafos dirigidos. Además, existen extensiones RDF diseñadas para representar de una manera formal bases de conocimiento y bases de datos terminológicas, que pueden ser utilizadas para representar recursos léxico-semánticos.

[2] Aunque muchas veces los esquemas de anotación requieren de librerías especializadas que interpreten la información representada.

[3] `https://www.w3.org/TR/json-ld`

Niveles de anotación

Los diferentes tipos de anotaciones lingüísticas se suelen agrupar en niveles según los diferentes procesos y herramientas PLN que las generan (ver sección 2.1). Estos niveles de anotación tienen muchas veces una estructura jerárquica, y anotaciones de niveles superiores hacen referencia muchas veces a anotaciones de niveles inferiores. La anotación de *tokens* y la *segmentación de frases/párrafos* es el nivel más bajo de anotación y todos los niveles de análisis subsiguientes suelen referirse a ésta. Usualmente, la anotación de tokens consiste en anclar el token utilizando las posiciones en el documento. A veces, se asocia cierta información al token tal como si es el comienzo de una frase, si es un número, etc. *La anotación morfosintáctica* suele asociar uno o más tokens (en el caso de unidades multi-palabra) con el análisis morfosintáctico correspondiente. Los análisis a veces son muy sencillos (lema, categoría principal), representados en forma de pares atributo/valor. A veces el análisis morfosintáctico es muy complejo, y requiere de formalismos como las estructuras de rasgos para su correcta representación.

A nivel sintáctico, las anotaciones de *chunking* consisten usualmente en una agrupación de unidades léxicas, mientras que las *dependencias sintácticas* son anotaciones entre pares de unidades léxicas, que se asocian con el tipo de relación entre la palabra dependiente y el núcleo. El caso más complejo de anotación sintáctica es sin duda el *análisis de constituyentes*, representado en forma de árboles de derivación, compuestos de nodos no terminales, nodos terminales (las palabras), núcleos de los constituyentes, y arcos entre nodos.

A nivel semántico, las anotaciones de *sentidos* consisten en asociar las unidades léxicas (palabras o unidades multi-palabras) con acepciones perteneciente a un repositorio pre-establecido. Típicamente, sólo se etiquetan las palabras de las llamadas categorías abiertas (sustantivos, verbos, adjetivos y adverbios) aunque categorías como preposiciones, artículos o pronombres también tengan una semántica definida. En cuanto a los repositorios de acepciones, *WordNet* [Mil+91] es quizá el repositorio más utilizado, pero a veces se utilizan también diccionarios en formato electrónico. Las anotaciones de *entidades nombradas* consisten en asociar una o más unidades léxicas con la categoría correspondiente (persona, lugar, organización, etc.). En el caso de la desambiguación, también se asocia la entidad con un elemento de una base de conocimiento externa (*Wikipedia*, *Dbpedia*, etc.). Virtualmente todas las anotaciones de expresiones temporales se adhieren, de una u otra forma, al estándar *TimeML* [Pus+10a] para el etiquetado de eventos y expresiones temporales.

Por último, las anotaciones de *correferencia* suelen identificar conjuntos de unidades léxicas y entidades nombradas que comparten el mismo referente.

3.3 Estándares para la representación de la información lingüística

Uno de los problemas más importantes que limita la interoperabilidad y reusabilidad de herramientas y recursos PLN es la heterogeneidad de las anotaciones lingüísticas. Modelos de anotación como *LAF*, *UIMA*, *TIPSTER*, etc. (ver sección 3.4) ofrecen una arquitectura común para representar las anotaciones, independiente de la aplicación PLN, dando una solución así a la llamada interoperabilidad estructural. Pero como hemos visto anteriormente, una integración real de componentes PLN requiere también que el significado de las etiquetas usadas en la anotación lingüística sea también compartida,

la llamada *interoperabilidad conceptual*. Por ejemplo, un analizador morfosintáctico puede anotar la palabra "llegó" con una etiqueta que represente el pretérito indefinido, tercera persona, y los subsiguientes módulos PLN deben conocer el significado de estas anotaciones. La comunidad PLN ha desarrollado diferentes repositorios terminológicos para anotaciones PLN, que ofrecen una suerte de terminología común para representar las anotaciones. Entre ellas, cabe destacar las siguientes.

3.3.1 Etiquetas Penn TreeBank

Las etiquetas de *Penn Treebank*[4] son un conjunto de etiquetas orientadas al análisis morfológico y morfosintáctico para el inglés, y describen las principales categorías gramaticales de las palabras. Las etiquetas de *Penn Treebank* representan un estándar *de facto* en muchas herramientas PLN que trabajan en inglés. Así, herramientas PLN como *CoreNLP* (ver sección 4.1) o *Freeling* para el inglés (sección 4.5) utilizan estas categorías para el etiquetado morfosintáctico. Además, existen multitud de corpus anotados utilizando este conjunto de etiquetas.

3.3.2 Etiquetas PAROLE y EAGLES

El proyecto *EAGLES* [Usz+96] fue pionero en la identificación de los problemas de interoperabilidad entre plataformas y recursos PLN. Al contrario que las etiquetas *Penn Treebank*, orientadas básicamente para el inglés, las etiquetas *EAGLES* fueron diseñadas específicamente para representar fenómenos lingüísticos en varios idiomas (básicamente europeos). Las recomendaciones *EAGLES* para la codificación de etiquetas morfosintácticas se convirtieron en un estándar de facto, y fueron ampliadas por proyectos posteriores. Por ejemplo, en el proyecto *PAROLE* [Rui+98] se ampliaron las etiquetas *EAGLES* para cubrir 12 lenguas europeas, y se diseñaron guías de anotación para el etiquetado de corpus textuales en esas lenguas.

3.3.3 Universal Dependencies

El proyecto *Universal Dependencies*[5] (UD) es un intento impulsado por *Google* de proveer etiquetas sintácticas unificadas (incluyendo las etiquetas morfosintácticas) para el etiquetado consistente de corpus a nivel sintáctico (los llamados *treebanks*) siguiendo el formalismo del análisis de dependencias sintácticas. UD está suscitando mucho interés en la comunidad científica, y muchos grupos lo están adoptando como estándar para el etiquetado morfosintáctico. Actualmente existen corpus anotados sintácticamente[6] utilizando las etiquetas UD en más de 40 idiomas, incluyendo muchos de los idiomas europeos e idiomas como el coreano, japonés o chino.

3.3.4 Data Category Registry (ISOcat)

El registro de categorías *ISOcat*[7] (*ISO TC37 Data Category Registry*) fue creado en 2008 con el objetivo de lograr una interoperabilidad real entre aplicaciones y recursos PLN

[4] `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`

[5] `http://universaldependencies.org/`

[6] Los corpus UD están anotados utilizando el formato *CoNLL-U*, una variante del formato *CoNLL* (ver sección 3.6.2)

[7] `http://www.isocat.org/`

[Kem+08]. El objetivo de *ISOcat* es el de proveer un conjunto reusables de categorías de datos para el análisis PLN a diferentes niveles. Las categorías están organizadas en niveles de análisis y cada una de ellas contiene también metainformación asociada como una descripción, lengua de aplicabilidad, ejemplos de uso, etc. En la actualidad *ISOcat* contiene una gran variedad de categorías para diferentes niveles de análisis, tales como morfosintaxis, sintaxis, semántica, etc.

3.3.5 GOLD

La ontología *GOLD* [FL03] ofrece una representación formal de las categorías y relaciones más básicas (llamadas "átomos") utilizadas en la descripción del lenguaje humano. Enmarcada dentro del campo de la lingüística descriptiva, *GOLD* intenta capturar el conocimiento de expertos lingüistas y puede considerarse, así, como un intento de codificar el conocimiento general en la materia. *GOLD* se alinea con los principios básicos de la web semántica y ofrece una ontología que pretende facilitar el razonamiento automático sobre datos lingüísticos y ayudar en establecer los conceptos básicos sobre los cuales se puedan llevar a cabo búsquedas inteligentes.

3.3.6 OLiA

La ontología *OLiA* (*Ontologies of Linguistic Annotation*) es un repositorio de categorías utilizado para anotar corpus[8]. Proveniente del área de la web semántica, *OLiA* es una iniciativa para integrar diferentes repositorios de etiquetas para describir información lingüística en diversos niveles. Las categorías están formalizadas por medio la lógica de descripciones OWL, lo cual permite declarar interrelaciones complejas entre categorías más allá de relaciones 1:1. Así, el objetivo fundamental de *OLiA* es proveer de un repositorio central de categorías lingüísticas. El repositorio es independiente de las aplicaciones concretas y realiza la función de mediador entre esquemas terminológicos definidos por diferentes sistemas y comunidades, tales como *GOLD* e *ISOcat*.

Actualmente *OLiA* contiene más de una treintena de modelos de anotación en más de 65 lenguajes, que cubren diferentes niveles de análisis como la morfología, morfosintaxis, sintaxis, semántica léxica y elementos pragmáticos como la anotación anafórica. No obstante, no existen corpus que estén anotados utilizando directamente etiquetas *OLiA*.

3.4 Formatos: modelos de anotación

En esta sección describiremos los modelos de anotación más utilizados actualmente, y cuyo objetivo es ofrecer una solución general para la anotación lingüística y dar solución a los problemas inherentes de este tipo de anotación como la ambigüedad, etc. Nos centraremos en aquellos modelos que tienen una vocación de estándar, en el sentido de que fueron diseñados con el objetivo de ser generales y no orientados a ninguna aplicación específica.

3.4.1 TIPSTER

TIPSTER es una arquitectura para el anotado general de textos [Gri96]. Principalmente orientado a facilitar el análisis de textos dentro de las agencias gubernamentales estadouni-

[8] `http://www.acoli.informatik.uni-frankfurt.de/resources/olia/`

Texto original				
Cyndi savored the soup.				
Anotaciones				
Id	Type	Span Start	Span End	Attributes
1	token	0	5	pos=NP
2	token	6	13	pos=VBD
3	token	14	17	pos=DT
4	token	18	22	pos=NN
5	token	22	23	
6	name	0	5	name_type=person
7	sentence	0	23	constituents=[1],[2],[3],[4],[5]
8	parse	0	5	symbol=NP constituents= [1]
9	parse	14	22	symbol=NP constituents=[3],[4]
10	parse	6	22	symbol=VP constituents=[2],[9]
11	parse	0	22	symbol=S constituents=[8],[10]

Figura 3.3: Anotaciones en TIPSTER.

denses, *TIPSTER* ofrece una arquitectura neutra para la gestión de anotaciones lingüísticas orientada a la anotación de grandes cantidades de textos. *TIPSTER* fue originalmente diseñada dentro del proyecto de mismo nombre (en el marco de los proyectos estadounidenses financiados por *DARPA*) como formato general de anotaciones con el objetivo de mejorar sistemas de búsqueda y extracción de información.

La arquitectura *TIPSTER* adopta el modo *stand-off* a la hora de anotar textos. Por una parte, la arquitectura mantiene el documento original intacto, y todas las anotaciones se efectúan utilizando *offset* al documento original. En el cuadro 3.3 se puede ver un ejemplo de la anotación de una frase[9]. Como se puede observar, las anotaciones pueden organizarse de forma jerárquica; las líneas 1 a 6 contienen anotaciones de tipo morfosintáctico que representan el tokenizado básico y la categoría gramatical de las palabras. Cada anotación está *anclada* a una porción del texto original por medio de *offsets* que marcan el comienzo y el final de la propia anotación. La línea 6 representa una entidad nombrada, de nuevo por medio de *offsets*. A partir de la línea 7 se representan anotaciones de tipo sintáctico, que están basadas, a su vez, en las anotaciones morfosintácticas definidas anteriormente.

La plataforma de procesamiento *GATE* (véase sección 4.3) adopta el modelo *TIPSTER* para representar las anotaciones, aunque lo hace de una forma algo distinta al formato original. La figura 3.4 muestra un ejemplo de una serialización XML del formato de anotaciones utilizado en *GATE*. La diferencia principal entre *TIPSTER* y el formato *GATE* es que este último no utiliza *offsets* para anclar las anotaciones. En lugar de eso, *GATE* inserta elementos `Node` dentro del texto original (normalmente representando los diferentes *tokens*) y el resto de anotaciones hacen referencia a estos nodos en el texto.

[9]El cuadro esta extraído de [Gri96].

```
<TextWithNodes><Node id="0"/>Seven <Node id="6"/>UK<Node id="8"/>
airlines including <Node id="28"/>...</TextWithNodes>
<AnnotationSet Name="Key" >
  <Annotation Id="43" Type="Organization" StartNode="1367" EndNode="1381">
    <Feature>
      <Name className="java.lang.String">rule1</Name>
      <Value className="java.lang.String">OrgXBase</Value>
    </Feature>
</AnnotationSet>
<AnnotationSet Name="Original markups" >
  <Annotation Id="79" Type="p" StartNode="938" EndNode="1127">
  </Annotation>
  <Annotation Id="112" Type="TEXT" StartNode="0" EndNode="2707">
  </Annotation>
</AnnotationSet>
```

Figura 3.4: Serialización XML del formato de anotaciones utilizado en GATE.

3.4.2 UIMA CAS

CAS (*Common Analysis Structure*) es el formalismo utilizado por la plataforma *UIMA*[10] para representar anotaciones. Las anotaciones en *UIMA* se definen por medio de las llamadas estructuras de rasgos (*feature structures*). Así, una anotación es simplemente una estructura de rasgos de un tipo determinado que está asociada (anclada) a una porción del texto que se está analizando (aunque también puede referirse a todo el documento en su conjunto). Todas las estructuras de rasgos en *UIMA* están representadas siguiendo el formalismo *CAS*. Así, *CAS* es la estructura de datos principal por el que los componentes *UIMA* se comunican entre ellos: todos los componentes (los llamados *Analysis Engines*) leen y generan anotaciones siguiendo el formalismo *CAS*.

Las anotaciones en *CAS* son de tipo *stand-off*, es decir, las anotaciones se mantienen de forma separada del documento. Un *CAS* encapsula estructuras de rasgos que están organizadas en vistas. Cada vista tiene un sujeto de análisis preciso (el texto que se está anotando), el resultado del análisis (metadatos que describen el sujeto de análisis) e índices a los resultados de los análisis.

Al ser un modelo de anotación, *CAS* no especifica el tipo de estructuras de rasgos a utilizar, ni su semántica precisa. Cada aplicación debe definir de antemano los tipos de anotaciones que va a utilizar por medio del llamado descriptor del sistema de tipos (*Type System Descriptor*). Esta información se describe de forma declarativa, y *UIMA* genera automáticamente los objetos *CAS* correspondientes que pueden compartirse entre los módulos. La plataforma *UIMA* proporciona al desarrollador de aplicaciones una interfaz para tratar directamente con objetos *CAS* a partir de la declaración de tipos descritos en el descriptor del sistema de tipos. De esta manera, el programador de módulos UIMA puede abstraerse del nivel físico de las anotaciones, y trabajar directamente en el nivel lógico.

El formalismo *CAS* está adaptado para trabajar de forma distribuida con grandes cantidades de datos [Eps+12]. Uno de los factores importantes a considerar cuando se distribuye el trabajo entre varios nodos de procesamiento es el tiempo empleado en serializar/de-serializar las anotaciones de los documentos. Una forma de aliviar este problema es que cada nodo de procesamiento reciba exclusivamente la información que necesite para realizar su tarea. En *UIMA*, esto se consigue creando niveles jerárquicos de anotaciones *CAS* por medio de los llamados multiplicadores *CAS* (*CAS Multiplier*,

[10] `https://uima.apache.org/`

[Eps+12]).

3.4.3 LAF (*Linguistic Annotation Framework*)

El objetivo de *LAF* es establecer un estándar definitivo basado en otros estándares de facto como *TEI*, *CES* (*Corpus Encoding Standard*) y su sucesor *XCES*. *LAF* es un marco para la representación de anotaciones lingüísticas diversas que incluye un modelo abstracto de datos de propósito general para las anotaciones y un formato de serialización XML denominado *GrAF* (*Graph Annotation Format*, [IS07]), el cual sirve como pivote para el mapeo entre formatos definidos por el usuario. *GrAF* viene acompañado de una librería[11] que ofrece una API para trabajar con documentos anotados siguiendo este formato.

El modelo de datos propuesto por LAF consta de tres partes:

1. anclajes que definen regiones como referencias a posiciones a los datos primarios y que constituyen los datos que deben ser anotados
2. un grafo formado por nodos, aristas y enlaces a las regiones definidas anteriormente
3. una estructura para la anotación que consta de un grafo dirigido que referencia a regiones o a otras anotaciones, y cuyos nodos se asocian con estructuras de rasgos que representan el contenido lingüístico

Para representar el contenido lingüístico, *LAF* se basa en *ISOcat* (ver subsección 3.3.4), una implementación de la ISO 12620:2009 desarrollada por ISO/TC 37/SC 3.

Cualquier recurso lingüístico conforme con *LAF* consta de:

- los datos primarios (texto, audio, video, etc.)
- al menos un documento con los anclajes definidos por la segmentación de los datos primarios en regiones analizables
- varios documentos con anotaciones en forma de nodos, aristas y estructuras de rasgos
- un conjunto de ficheros con metadatos especificados de manera parecida a como se hace en *CES*.

3.4.4 NXT

NXT[12] (*NITE XML Toolkit*) es un conjunto de herramientas para trabajar con datos multimodales, incluyendo datos textuales y corpus hablados [Car+03]. *NXT* organiza las anotaciones siguiendo una estructura general de grafos, donde tanto los nodos y los arcos pertenecen a diferentes tipos. Utilizando esta estructura común, *NXT* es capaz de anotar documentos multimodales en varios niveles de análisis de forma jerárquica.

3.4.5 NIF (*NLP Interchange Format*)

NIF es un formato general para representar las anotaciones lingüísticas que se adhiere al paradigma de los datos enlazados (*Linked Data*). Las anotaciones en *NIF* se representan en RDF (ver sección 3.2) y, consecuentemente, todos los elementos que conforman las anotaciones *NIF* son recursos RDF representados por identificadores globales unívocos (URIs), incluyendo las anclas a documentos, las relaciones y los elementos de anotación. Estos dos últimos se refieren usualmente a elementos estándar como los descritos en la sección 3.3.

[11] `https://sourceforge.net/projects/iso-graf`
[12] `http://groups.inf.ed.ac.uk/nxt/`

```
<http://freme-project.eu/#char=0,4>
        a                       nif:RFC5147String , nif:Word ;
        nif:anchorOf            "Mrs." ;
        nif:beginIndex          "0" ;
        nif:endIndex            "4" ;
        nif:nextWord            <http://freme-project.eu/#char=5,12> ;
        nif:referenceContext    <http://freme-project.eu/#char=0,217> ;
        nif:sentence            <http://freme-project.eu/#char=0,160> ;
        itsrdf:taIdentRef       <http://dbpedia.org/resource/Mrs.> .

<http://freme-project.eu/#char=5,12>
        a                       nif:Word , nif:RFC5147String ;
        nif:anchorOf            "Clinton" ;
        nif:beginIndex          "5" ;
        nif:endIndex            "12" ;
        nif:nextWord            <http://freme-project.eu/#char=13,15> ;
        nif:previousWord        <http://freme-project.eu/#char=0,4> ;
        nif:referenceContext    <http://freme-project.eu/#char=0,217> ;
        nif:sentence            <http://freme-project.eu/#char=0,160> ;
        itsrdf:taIdentRef       <http://dbpedia.org/resource/Bill_Clinton> .

<http://freme-project.eu/#char=51,61>
        a                       nif:RFC5147String , nif:Word ;
        nif:anchorOf            "Department" ;
        nif:beginIndex          "51" ;
        nif:endIndex            "61" ;
        nif:nextWord            <http://freme-project.eu/#char=62,71> ;
        nif:previousWord        <http://freme-project.eu/#char=45,50> ;
        nif:referenceContext    <http://freme-project.eu/#char=0,217> ;
        nif:sentence            <http://freme-project.eu/#char=0,160> ;
        itsrdf:taIdentRef       <http://dbpedia.org/resource/Departments_of_France> .
```

Figura 3.5: Ejemplo de anotaciones NIF para desambiguar entidades nombradas.

NIF sufre de una cierta sobrecarga a la hora de representar las anotaciones, debido principalmente al inherente modelo de datos utilizado en RDF. Por ejemplo, asignar atributos a las relaciones es muchas veces complicado, y requiere aplicar el método de reificación sobre la relación, es decir, convertir la relación en una clase más a la que se pueda asignar atributos. Por otro lado, RDF carece de estructuras de datos básicas como la secuencia o el conjunto. Estas estructuras son necesarias en la anotación lingüística, ya que el texto está formado por secuencias de palabras y el orden de las palabras cumple un papel fundamental. Para solucionar este problema, *NIF* necesita utilizar elementos adicionales para representar implícitamente estas anotaciones.

La figura 3.5 muestra un extracto de anotaciones *NIF* para la desambiguación de entidades nombradas. Las palabras se referencian utilizando URIs que describen unívocamente la posición de éstas en el documento. Las relaciones `nif:nextWord` o `nif:previousWord` en la figura representan de forma implícita la secuencia de palabras del texto original. Nótese también que el orden de las tripletas no coincide con el orden de las palabras en el documento.

3.5 Formatos: esquemas de anotación

En esta sección describiremos los esquemas de anotación más relevantes, incluyendo estándares ISO. Recordemos que los esquemas de anotación especifican los elementos concretos a utilizar para representar las anotaciones lingüísticas en varios niveles. Entre los

esquemas que describiremos, algunos tienen un carácter general y cubren muchos niveles de anotación (p.ej., *NAF* o *FoLIA*), mientras que otros esquemas están específicamente definidos para representar anotaciones de un cierto nivel lingüístico, como *SynAF* para anotaciones sintácticas, o *SemAF* para anotaciones semánticas.

3.5.1 NAF (*NLP Annotation Format*)

El formato *NAF* (*NLP Annotation Format*, [Fok+14]) fue desarrollado dentro del proyecto *NewsReader*[13] como formato de intercambio entre múltiples procesadores lingüísticos. Basado en XML, *NAF* adopta los principios básicos de *LAF*, entre los que cabe destacar la máxima flexibilidad, eficiencia de proceso y reusabilidad. *NAF* es un formato *stand-off* multicapa donde cada procesador incluye de forma incremental nuevas capas sobre el documento con las anotaciones pertinentes. *NAF* incluye una cabecera con meta-información del documento original, así como una traza con el registro de los procesadores PLN empleados. La figura 3.2 (en la página 21) muestra un ejemplo de anotación *NAF*.

NAF está diseñado para poder ser usado en entornos distribuídos y paralelos. Así, los módulos típicamente añaden capas al documento sin modificar capas pre-existentes. Esto permite que *NAF* pueda ser utilizado tanto en entornos de procesamiento por lotes (*batch*) como en procesamiento en línea (*streaming*, véase capítulo 6), y que múltiples módulos PLN pueden analizar el mismo documento de forma concurrente.

Como hemos dicho anteriormente, *NAF* está basado en XML, y su especificación incluye un esquema *DTD* que describe la sintaxis concreta de los elementos. No obstante, existe también una serialización *NAF* utilizando el formato *JSON*. Esta serialización es utilizada en [Bel+16] en un entorno distribuido para almacenar grandes cantidades de documentos de forma nativa en una base de datos *MongoDB*.

Las especificación del formato *NAF* son accesibles públicamente[14]. Además, *NAF* viene acompañada de una librería Java llamada *kaflib*[15], que ofrece una API para gestionar las anotaciones *NAF*.

3.5.2 FoLiA (*Format for Linguistic Annotation*)

FoLiA es el formato de anotación desarrollado por la Universidad de Radboud en Nimega para dar solución a los problemas de intercambio de datos entre aplicaciones lingüísticas [GR13]. Este formato es el utilizado para la plataforma de flujo de trabajos *TTNWW*. *FoLiA* es un formato mixto que representa un documento anotado como un sólo fichero XML con toda las anotaciones incluidas. Este formato está diseñado para ser legible y expresivo dejando de lado su eficiencia de procesamiento y usa varias marcas XML en función del tipo de anotación: estructura, *token*, *span* y *subtoken*. La figura 3.6 muestra un extracto de un documento anotado en *FoLiA*.

3.5.3 TO2 (*Travelling Object 2*)

El *Travelling Object 2* (TO2, [PB11]) es el formato de anotación *stand-off* utilizado en el proyecto *Panacea* (ver sección 7.4). *TO2* está basado en el modelo *LAF* serializado en *GrAF*, e integra el *Travelling Object 1* (*TO1*) para los datos primarios (i.e., la representación

[13] `http://www.newsreader-project.eu`
[14] `https://github.com/newsreader/NAF`
[15] `https://github.com/ixa-ehu/kaflib`

```
<FoLiA ... >
  <metadata type="native">
    <annotations>
     <token-annotation annotator="ilktok" annotatortype="auto" />
     ...
    </annotations>
  </metadata>
  <text xml:id="WR-P-E-J-0000000001.text"><lang class="nl"/>
    <div xml:id="WR-P-E-J-0000000001.div0.1" class="chapter">
      <p xml:id="WR-P-E-J-0000000001.p.1" class="firstparagraph">
        <s xml:id="WR-P-E-J-0000000001.p.1.s.1">
          <w xml:id="WR-P-E-J-0000000001.p.1.s.1.w.1">
            <t>Stemma</t>
            <pos class="N(eigen,ev,basis,zijd,stan)"/>
            <lemma class="Stemma"/>
          </w>
                  ...
          <entities>
            <entity class="ander_woord" set="mwu-set">
              <wref id="WR-P-E-J-0000000001.p.1.s.1.w.4" t="ander"/>
              <wref id="WR-P-E-J-0000000001.p.1.s.1.w.5" t="woord"/>
            </entity>
          </entities>
          <syntax>...</syntax>
          <dependencies>...</dependencies>
          <chunking>...</chunking>
          <timing>...</timing>
        </s>
      </p>
    </div>
  </text>
</FoLiA>
```

Figura 3.6: Extracto de un documento anotado según el formato FoLiA

electrónica de los datos fuente), inspirado en *XCES*. El *TO2* está compuesto de varios documentos:

- uno o más documentos con datos primarios
- uno o más documentos con referencias a los datos primarios y que proveen la segmentación básica para otras anotaciones
- documentos con anotaciones lingüísticas que hacen referencia a la segmentación básica o a otros documentos con anotaciones
- documentos con las cabeceras asociadas a cada documento con datos primarios o con anotaciones.

Los documentos con anotaciones lingüísticas son el resultado del procesamiento de los datos primarios o de sus anotaciones, en el caso de anotaciones por niveles. En *PANACEA* cada nivel de análisis requiere un documento de anotación independiente. Siguiendo las recomendaciones de *LAF/GrAF*, todos los documentos con datos primarios llevan asociados ficheros XML independientes con una cabecera inspirada en *CES*. Muchos de los documentos en *Panacea* son documentos HTML, su estructura (*layout*) y formato (*rendering*) están representados en otro documento.

3.5.4 MAF (*The Morpho-Syntactic Annotation Framework*)

MAF tiene como objetivo la estandarización de la anotación morfosintáctica de datos primarios [CL05]. Para ello, MAF provee tanto el modelo de datos como el formato de anotación *stand-off* para representar la segmentación y la anotación de datos primarios

en dos niveles. En un primer nivel, los datos primarios son segmentados como tokens que son usados como base para definir el segundo nivel, o nivel de las palabras (*word forms*), mediante relaciones *n*:*m*. De manera similar a LAF, tanto los tokens como las palabras pueden organizarse como un grafo dirigido acíclico para representar la ambiguedad en la segmentación estructural y morfológica. A su vez, las palabras llevan asignadas anotaciones en forma de estructuras de rasgos en las que se describen sus propiedades. El etiquetario para la anotación morfosintáctica no está especificado por el estándar pero recomienda que se base en las categorías descritas en el registro de categorías *ISOcat*.

3.5.5 SynAF (*The Syntactic Annotation Framework*)

SynAF es un estándar que complementa a *MAF* y que describe las relaciones sintácticas entre palabras y su agrupación en sintagmas y oraciones, es decir, la estructura de dependencias y constituyentes [Dec06]. *SynAF* define un metamodelo para representar esta información sintáctica y ofrece un conjunto de categorías de datos basados en *ISOcat*.

3.5.6 SemAF (*Semantic annotation framework*)

SemAF son de hecho varios estándares para la anotación semántica. Por un lado *SemAF-Time* [Pus+10b] especifica un lenguaje de marcación XML, denominado *ISO-TimeML*, para la extracción y representación de información temporal y para facilitar el intercambio de esta información, tanto entre sistemas de procesamiento como entre esquemas de representación de información temporal.

DiAML es el lenguaje de marcación estándar para la anotación de dialogos. El estándar define también el método para la segmentación de diálogos en unidades semánticas y las categorías de funciones comunicativas, las dimensiones del análisis del diálogo y los principios y guías para extender estas categorías [Har+12].

Otros aspectos de la anotación semántica están o estarán cubiertos por *SemAF-NE* (entidades nombradas), *SemAF-SR* (roles semánticos), *SemAF-DS* (estructura discursiva) e *ISOspace* (información espacial).

3.5.7 TCF (*Text Corpus Format*)

TCF es el formato usado por el participante alemán de *CLARIN* en su plataforma *Weblicht*[16]. Es un formato bastante verboso ya que usa marcas XML para las anotaciones que representa. No usa referencias a posiciones de caracteres pero usa referencias a identificadores de token cuando se anclan anotaciones de niveles de análisis superiores.

3.6 Formatos de anotación *ad hoc*

Llamamos formatos de anotación *ad hoc* a aquellos formatos utilizados por aplicaciones y herramientas PLN para representar sus resultados en forma de anotaciones lingüísticas. Estos formatos están orientados a cubrir las necesidades concretas de la aplicación y normalmente no tienen vocación de estándar, en el sentido de que no están diseñados para cumplir los objetivos de reutilización, interoperabilidad entre componentes, soporte de múltiples idiomas, etc. Aun así, muchas de las herramientas PLN más importantes definen sus propios formatos para representar la anotaciones.

[16] `http://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/The_TCF_Format`

```
U.N.          NNP  I-NP  I-ORG
official      NN   I-NP  O
Ekeus         NNP  I-NP  I-PER
heads         VBZ  I-VP  O
for           IN   I-PP  O
Baghdad       NNP  I-NP  I-LOC
.             .    O     O
```

Figura 3.7: Ejemplo del formato CoNLL (2003)

Los formatos *ad hoc* específicos pueden ser muchas veces traducidos a formatos estándar, y existen pequeños programas o *scripts* para llevar a cabo estas conversiones. Por el contrario, convertir anotaciones representadas en formatos estándar a estos formatos específicos es a menudo imposible.

3.6.1 CoNLL

El congreso *Computational Natural Language Learning*, más conocido con las siglas *CoNLL*[17] tiene como objetivo estudiar la aplicación de métodos de aprendizaje automático (*machine learning*) en el área del PLN. Es un congreso que se celebra anualmente, y desde el año 1999 organiza una tareas conjuntas que analiza una problemática concreta. Los participantes en la tarea reciben un conjunto de entrenamiento y de *test*, y deben aplicar técnicas de aprendizaje automático para resolver la tarea. A lo largo de su historia, *CoNLL* ha organizado tareas centradas en problemas tales como el *chunking* o sintaxis superficial, análisis sintácticos de dependencias o el reconocimiento de entidades nombradas, etc.

Los datos para las tareas conjuntas *CoNLL* vienen en un formato concreto, que se ha venido a denominar el formato *CoNLL*. No existe un único formato *CoNLL*, ya que el formato de cada tarea es diferente al anterior. Existe no obstante una serie de características comunes en todos los formatos *CoNLL*, y la principal de ellas, y lo que la diferencia del resto, es el hecho de ser un formato tabulado. En el formato *CoNLL* cada línea, que representa una palabra o token, está compuesta por un número fijo de campos, separados entre sí por un tabulador. Estos campos codifican diferente información de la palabra, por ejemplo su forma, lema, categoría gramatical, etc.

Por ejemplo, la figura 3.7 muestra un ejemplo de una sentencia anotada siguiendo el formato *CoNLL 2003*. Cada línea contiene cuatro columnas, separadas por un tabulador. Las columnas corresponden con la forma del token, la categoría gramatical y dos etiquetas sintácticas que indican si la palabra pertenece a un *chunk* o entidad nombrada, respectivamente.

El formato *CoNNL-U* es un formato *CoNNL* utilizado por el proyecto *Universal Dependencies* de Google (ser sección 3.3.3) para etiquetar textos en varios idiomas con información morfosintáctica y sintáctica.

3.6.2 Stanford CoreNLP

CoreNLP de la universidad de Stanford es un conjunto de módulos PLN para el procesamiento de diferentes idiomas (ver sección 4.1). *CoreNLP* utiliza su propio formato XML para representar las anotaciones que genera. *CoreNLP* es un formato en línea y la

[17] http://www.conll.org/

	Modelo/Esquema	Etiquetas	Librería/API	Metadatos	Versionado	Multimedia	Stand-off	Serialización
CAS	M	Cualquiera	✓	✓	✓	X	✓	XML, otros
FoLiA	E	Propios	✓	✓	✓	X	X	XML
Graf	M	ISOcat	✓	X	X	X	✓	XML
MAF	E	ISOcat	X	X	X	X	✓	XML
NAF	E	Propios	✓	✓	✓	X	✓	XML, JSON
NIF	M	OLia	✓	X	X	X	✓	RDF
NXT	M	Cualquiera	✓	✓	✓	✓	✓	XML
SemAF	E	ISOcat	X	X	X	X	✓	XML
SynAF	E	ISOcat	X	X	X	X	✓	XML
TCF	E	Propios	✓	X	X	X	✓	XML
TIPSTER	M	Cualquiera	✓	✓	✓	X	✓	XML, otros
TO2	E	ISOcat	X	✓	X	X	✓	XML

Tabla 3.1: Tabla comparativa con diferentes características en formatos de anotación

información está a veces repetida en diferentes niveles (por ejemplo, el lema de los *tokens* se repite en varios niveles de anotación).

3.6.3 FreeLing

Freeling es una colección de herramientas para el procesamiento PLN y análisis de textos (ver sección 4.5). FreeLing funciona como una librería, y dispone de una API para acceder a todas sus funcionalidades de análisis. *FreeLing* dispone de diversos módulos *back-end* que permiten representar su salida utilizando diversos formatos de serialización (XML, JSON, tabulado, etc.). La mayoría de las etiquetas en FreeLing se adhieren al estándar *EAGLES*, y las etiquetas para el inglés están representadas utilizando las etiquetas del *Penn TreeBank*.

3.7 Tabla comparativa

La tabla 4.1 muestra una comparación de diferentes formatos de anotación y resalta diferentes características de cada uno. Las características que se han tenido en cuenta en la tabla son los siguientes:

- **Modelo/Esquema**: como hemos visto, ciertos formatos ofrecen una solución genérica para la representación de las anotaciones (el modelo de anotación) pero no definen expresamente qué esquema de anotación utilizar. Otros formatos en cambio sí definen los esquemas de anotación, es decir, el conjunto de elementos a utilizar para representar la información en diferentes niveles de análisis. Los posibles valores son M para indicar que se trata de un modelo de anotación, o E si es un esquema de anotación.
- **Etiquetas**: si los formatos se adhieren a algún estándar conceptual para representar el conjunto de etiquetas (ver sección 3.3).
- **Librería/API**: si el formato de anotación viene acompañado de una librería o API para poder gestionar documentos anotados.
- **Metadatos**: si el formato ofrece la posibilidad de anotar meta-información del documento, como el nombre del mismo, fecha, autor, etc.
- **Versionado**: unido al anterior punto, si el formato ofrece la posibilidad de especificar diferentes versiones en las anotaciones.
- **Multimedia**: si el formato permite anotar documentos multimedia, incluyendo, además del texto, audio/voz, imágenes, etc.

- **Stand-off**: si el formato es *stand-off* o en línea.
- **Formato serialización**: formatos posibles para serializar las anotaciones.

4. Herramientas PLN

En estos momentos, es posible acceder a un conjunto de paquetes de herramientas de amplia distribución destinadas al procesamiento lingüístico de carácter general. Estos paquetes de herramientas se ofrecen como código portable para su instalación y uso local, de tal forma que permiten el procesado PLN a diferentes niveles en varios idiomas. Los componentes de cada paquete tendrán un rendimiento distinto para cada idioma procesado. Así mismo, el resultado de cada componente será distinto según el dominio en el que se aplique el procesamiento. Entre los paquetes accesibles, destacan por sus funcionalidades *Stanford CoreNLP*, *Apache OpenNLP*, *GATE*, *NLTK*, *FreeLing* e *IXA pipes*.

En este capítulo describiremos estos paquetes, indicando, para cada uno, una serie de características:

- *Entorno de ejecución*: el lenguaje de programación y entorno de ejecución del paquete.
- *Licencia*: el tipo de licencia. En este estudio sólo consideramos los paquetes PLN de licencia libre.
- *Herramientas básicas*: el tipo de análisis PLN que el paquete realiza.
- *Soporte de idiomas*: En general todos lo paquetes pueden procesar textos en inglés, pero algunos de ellos también soportan otros idiomas, incluidos los idiomas cooficiales del estado español.
- *Formatos de salida*: el formato utilizado por la herramienta para representar las anotaciones lingüísticas.
- *Ciclo de vida*: en este apartado analizamos aspectos relativos al ciclo de vida de la herramienta, como por el ejemplo si permite entrenar nuevos modelos e integrarlos en el sistema.

4.1 Stanford CoreNLP

La distribución *CoreNLP* de Stanford incluye una serie de procesadores lingüísticos basados en aprendizaje automático, desarrollados en Java y diseñados para su uso en tubería (*pipeline*) ([Man+14]).

Sitio web

`http://stanfordnlp.github.io/CoreNLP/`

Versión actual

3.6.0 (9/12/2015)

Entorno de ejecución

Java 1.8+

Licencia

GNU General Public License v3+ (GPL)

Herramientas básicas

- Tokenizador.
- Segmentador de frases.
- Etiquetador morfosintáctico (*tagger*): asigna una etiqueta POS (*part-of-speech*) a cada token
- Lematizador.
- Reconocedor de entidades nombradas.
- Analizador sintáctico de constituyentes.
- Analizador sintáctico de dependencias.
- Sistema para la resolución de la correferencia.

Otras herramientas

- Análisis de sentimiento: asigna una valoración (positiva, negativa o neutra) de la polaridad de cada frase.
- Extracción de información abierta (módulo OpenIE): extrae las triplas sujeto-relación-objeto de las frases del texto.

Soporte de idiomas

La distribución básica de *CoreNLP* sólo proporciona los modelos para el análisis del inglés estándar (en), pero el motor es compatible con modelos para otros idiomas. Desde el sitio de *CoreNLP* se pueden descargar separadamente modelos lingüísticos para el análisis del chino (zh), francés (fr), alemán (de) y español (es), y modelos para géneros de inglés no estándar. Los modelos de Stanford para el árabe (ar) también se pueden usar con *CoreNLP*. Véase la Tabla 4.1 para una comparación del soporte de idiomas por herramienta.

Formatos de salida

CoreNLP ofrece diversos tipos de salidas del procesamiento:

Tabla 4.1: Soporte de idiomas por herramienta en *CoreNLP*

Herramienta por idioma	ar	zh	en	fr	de	es
Tokenizador	✓	✓	✓	✓		✓
Segmentador de frases	✓	✓	✓	✓	✓	✓
Etiquetador morfosintáctico	✓	✓	✓	✓	✓	✓
Lematizador			✓			
Entidades nombradas		✓	✓		✓	✓
Sintaxis de constituyentes	✓	✓	✓	✓	✓	✓
Sintaxis de dependencias		✓	✓	✓	✓	
Análisis de sentimiento			✓			
Resolución de la correferencia		✓	✓			
Extracción de información			✓			

- XML: Formato de salida predeterminado de *CoreNLP* (ver sección 3.6.2). Viene acompañado de una hoja de estilo *XSLT* para su visualización en un navegador.
- texto: Formato de texto *ad hoc* (legible para un ser humano)
- JSON
- *CoNLL* y *CoNLL-U* (ver sección 3.6.2)

Ciclo de vida

CoreNLP permite crear e incorporar nuevos módulos (anotadores) al procesamiento sin alterar el código principal de `StanfordCoreNLP.java`, simplemente extendiendo una de sus clases (`edu.stanford.nlp.pipeline.Annotator`).

Los modelos se distribuyen separadamente del código. Así mismo, *CoreNLP* permite reentrenar los módulos usando modelos para su procesamiento. Véase, por ejemplo, la instrucción para entrenar el módulo de análisis de sentimientos a partir de los corpus de entrenamiento `train.txt` y de desarrollo `dev.txt`:

```
java -cp "*" -mx8g edu.stanford.nlp.sentiment.SentimentTraining -numHid 25 \\
     -trainPath train.txt -devPath dev.txt -train -model model.ser.gz
```

Ejemplo de procesamiento: texto en inglés

Se muestra aquí como ejemplo de procesamiento de esta librería la instrucción para la ejecución y la salida por defecto en XML del análisis del texto «Stanford University is located in California. It is a great university, founded in 1891.» con los módulos disponibles para el inglés y el modelo para esta lengua en la distribución estándar de *CoreNLP*.

```
java -cp "*" -Xmx2g edu.stanford.nlp.pipeline.StanfordCoreNLP \\
     -annotators tokenize,ssplit,pos,lemma,ner,parse,dcoref,natlog,openie,sentiment \\
     -file input_corenlp_en.txt
```

```
<sentences>
  <sentence id="1" sentimentValue="1" sentiment="Negative">
```

```
      <tokens>
        <token id="1">
          <word>Stanford</word>
          <lemma>Stanford</lemma>
          <CharacterOffsetBegin>0</CharacterOffsetBegin>
          <CharacterOffsetEnd>8</CharacterOffsetEnd>
          <POS>NNP</POS>
          <NER>ORGANIZATION</NER>
          <Speaker>PER0</Speaker>
          <sentiment>Neutral</sentiment>
        </token> [...]
      </tokens>
      <parse>(ROOT
        (S
        (NP (NNP Stanford) (NNP University))
        (VP (VBZ is)
        (ADJP (JJ located) [...]
        </parse>
      <dependencies type="basic-dependencies">
        <dep type="root">
          <governor idx="0">ROOT</governor>
          <dependent idx="4">located</dependent>
        </dep>
        <dep type="compound">
          <governor idx="2">University</governor>
          <dependent idx="1">Stanford</dependent>
        </dep> [...]
      </dependencies>
      <openie> [...]
        <triple confidence="1,000">
          <subject begin="0" end="2">
            <text>Stanford University</text>
            <lemma>Stanford University</lemma>
          </subject>
          <relation begin="2" end="4">
            <text>is located in</text>
            <lemma>be located in</lemma>
          </relation>
          <object begin="5" end="6">
            <text>California</text>
            <lemma>California</lemma>
          </object>
        </triple>
      </openie>
    </sentence> [...]
  </sentences>
  <coreference>
    <coreference>
      <mention representative="true">
        <sentence>1</sentence>
        <start>1</start>
        <end>3</end>
        <head>2</head>
        <text>Stanford University</text>
      </mention>
      <mention>
        <sentence>2</sentence>
        <start>1</start>
        <end>2</end>
        <head>1</head>
        <text>It</text>
      </mention> [...]
    </coreference>
  </coreference>
```

Figura 4.1: Análisis de un texto en inglés con *CoreNLP*

Ejemplo de procesamiento: texto en español

A continuación, se ilustra el uso de la librería *CoreNLP* para el procesamiento del español con el análisis del texto «El actor británico Burt Kwouk falleció este martes en Londres a los 85 años.», con los módulos disponibles para el español y el modelo para esta lengua descargable desde el sitio de *CoreNLP*.

```
java -mx1g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLP \\
     -props StanfordCoreNLP-spanish.properties -annotators tokenize,ssplit,pos,ner,parse \\
     -file input_corenlp_es.txt
```

```
<sentences>
  <sentence id="1">
    <tokens>
      <token id="1">
        <word>El</word>
        <CharacterOffsetBegin>0</CharacterOffsetBegin>
        <CharacterOffsetEnd>2</CharacterOffsetEnd>
        <POS>da0000</POS>
        <NER>O</NER>
      </token>
      <token id="2">
        <word>actor</word>
        <CharacterOffsetBegin>3</CharacterOffsetBegin>
        <CharacterOffsetEnd>8</CharacterOffsetEnd>
        <POS>nc0s000</POS>
        <NER>O</NER>
      </token>
      <token id="3">
        <word>británico</word>
        <CharacterOffsetBegin>9</CharacterOffsetBegin>
        <CharacterOffsetEnd>18</CharacterOffsetEnd>
        <POS>aq0000</POS>
        <NER>O</NER>
      </token>
      <token id="4">
        <word>Burt</word>
        <CharacterOffsetBegin>19</CharacterOffsetBegin>
        <CharacterOffsetEnd>23</CharacterOffsetEnd>
        <POS>np00000</POS>
        <NER>PERS</NER>
      </token>
      <token id="5">
        <word>Kwouk</word>
        <CharacterOffsetBegin>24</CharacterOffsetBegin>
        <CharacterOffsetEnd>29</CharacterOffsetEnd>
        <POS>np00000</POS>
        <NER>PERS</NER>
      </token>
      <token id="6">
        <word>falleció</word>
        <CharacterOffsetBegin>30</CharacterOffsetBegin>
        <CharacterOffsetEnd>38</CharacterOffsetEnd>
        <POS>vmis000</POS>
        <NER>O</NER>
      </token>  [...]
    </tokens>
    <parse>(ROOT
      (sentence
      (sn
      (spec (da0000 El))
      (grup.nom (nc0s000 actor)
      (s.a
      (grup.a (aq0000 británico)))
      (sn
      (grup.nom (np00000 Burt) (np00000 Kwouk)))))
      (grup.verb (vmis000 falleció)) [...]
```

```
      </parse>
    </sentence>
  </sentences>
```

Figura 4.2: Análisis de un texto en español con CoreNLP

4.2 Apache OpenNLP

La biblioteca *OpenNLP* de Apache incluye un conjunto de herramientas para el procesamiento del lenguaje natural basadas en aprendizaje automático, desarrolladas en Java y diseñadas para su uso en tubería.

Sitio web

`https://opennlp.apache.org`

Versión actual

1.6.0 (13/7/2015)

Entorno de ejecución

Java 1.7+

Licencia

Apache License 2.0 (APL 2.0)

Herramientas básicas

- Segmentador de frases.
- Tokenizador.
- Etiquetador morfosintáctico.
- Reconocedor de entidades nombradas.
- Analizador sintáctico superficial.
- Analizador sintáctico de constituyentes.
- Sistema para la resolución de la correferencia.

Otras herramientas

- Clasificador de documentos: clasifica el texto en una serie de categorías predefinidas. Esta herramienta no cuenta con un modelo entrenado en la distribución de *OpenNLP*.

Soporte de idiomas

Existen modelos preentrenados en diferentes tareas para los idiomas danés (da), alemán (de), inglés (en), español (es), neerlandés (nl), portugués (pt) y sueco (se), distribuidos separadamente y descargables desde `http://opennlp.sourceforge.net/models.html`. Véase la Tabla 4.2 para una comparación del soporte de idiomas por herramienta.

Formatos de salida

Los procesadores de *OpenNLP* poseen un formato propio de salida legible para un ser humano formado por secuencias de cadenas de caracteres.

Tabla 4.2: Soporte de idiomas por herramienta en *OpenNLP*

Herramienta por idioma	da	nl	en	se	de	pt	es
Segmentador de frases	✓	✓	✓	✓	✓	✓	
Tokenizador	✓	✓	✓	✓	✓	✓	
Etiquetador morfosintáctico	✓	✓	✓	✓	✓	✓	✓
Entidades nombradas		✓	✓				✓
Sintaxis superficial			✓				
Sintaxis de constituyentes			✓				
Resolución de la correferencia			✓				

Ciclo de vida

Los modelos se distribuyen separadamente del código. Así mismo, *OpenNLP* permite reentrenar los módulos usando modelos para su procesamiento. Véase, por ejemplo, la instrucción empleada para entrenar el modelo para el análisis sintáctico de constituyentes del inglés a partir del corpus de entrenamiento `train.all`:

```
opennlp ParserTrainer -model en-parser-chunking.bin -parserType CHUNKING -lang en \\
       -head-rules head_rules -data train.all -encoding ISO-8859-1
```

Ejemplo de procesamiento: texto en inglés

Se muestra aquí, como ejemplo de procesamiento de esta librería, la instrucción para la ejecución y la salida del análisis sintáctico de constituyentes y de dependencias de un texto periodístico con los módulos disponibles para el inglés y los modelos para esta lengua en la distribución estándar de *OpenNLP*.

> Early on Saturday, a middle-aged Pashtun man used forged documents to cross from Iran into Pakistan. A few hours later, on a lonely stretch of highway, he was incinerated by an American drone. It is not exactly clear how the Americans tracked Mullah Akhtar Muhammad Mansour, leader of the Afghan Taliban, to a white sedan rattling across the arid expanse of Baluchistan Province. The United States picked up a mix of phone intercepts and tips from sources, American and European officials said, and there were reports that Pakistan also provided intelligence. President Obama described Mullah Mansour's death on Monday as an 'important milestone', but the strike was also an illustration of the tangled relationship between Washington and Islamabad.

Figura 4.3: Texto en inglés para analizar con *OpenNLP*

```
opennlp SentenceDetector en-sent.bin < input_opennlp_en.txt |
opennlp TokenizerME en-token.bin | opennlp POSTagger en-pos-maxent.bin |
opennlp ChunkerME en-chunker.bin
```

```
[ADVP Early_RB ] [PP on_IN ] [NP Saturday_NNP ] ,_, [NP a_DT middle-
    aged_JJ Pashtun_NNP man_NN ] [VP used_VBD ] [NP forged_VBN
    documents_NNS ] [VP to_TO cross_VB ] [PP from_IN ] [NP Iran_NNP ] [
    PP into_IN ] [NP Pakistan_NNP ] ._.
[NP A_DT few_JJ hours_NNS ] [ADVP later_RB ] ,_, [PP on_IN ] [NP a_DT
    lonely_JJ stretch_NN ] [PP of_IN ] [NP highway_NN ] ,_, [NP he_PRP
    ] [VP was_VBD incinerated_VBN ] [PP by_IN ] [NP an_DT American_JJ
    drone_NN ] ._.
[NP It_PRP ] [VP is_VBZ ] not_RB [ADVP exactly_RB clear_JJ ] [ADVP
    how_WRB ] [NP the_DT Americans_NNPS ] [VP tracked_VBD ] [NP
    Mullah_NNP Akhtar_NNP Muhammad_NNP Mansour_NNP ] ,_, [NP leader_NN
    ] [PP of_IN ] [NP the_DT Afghan_NNP Taliban_NNP ] ,_, [PP to_TO ] [
    NP a_DT white_JJ sedan_NN ] [VP rattling_VBG ] [PP across_IN ] [NP
    the_DT arid_JJ expanse_NN ] [PP of_IN ] [NP Baluchistan_NNP
    Province_NNP ] ._.
[NP The_DT United_NNP States_NNP ] [VP picked_VBD ] [PRT up_RP ] [NP
    a_DT mix_NN ] [PP of_IN ] [NP phone_NN intercepts_NNS ] and_CC [NP
    tips_NNS ] [PP from_IN ] [NP sources_NNS ] ,_, [NP American_JJ
    and_CC European_JJ officials_NNS ] [VP said_VBD ] ,_, and_CC [NP
    there_EX ] [VP were_VBD ] [NP reports_NNS ] [PP that_IN ] [NP
    Pakistan_NNP ] [ADVP also_RB ] [VP provided_VBD ] [NP
    intelligence_NN ] ._.
[NP President_NNP Obama_NNP ] [VP described_VBD ] [NP Mullah_NNP
    Mansour_NNP ] [NP 's_POS death_NN ] [PP on_IN ] [NP Monday_NNP ] [
    PP as_IN ] [NP an_DT 'important_JJ milestone_NN '_'' ] ,_, but_CC [
    NP the_DT strike_NN ] [VP was_VBD ] [ADVP also_RB ] [NP an_DT
    illustration_NN ] [PP of_IN ] [NP the_DT tangled_JJ relationship_NN
     ] [PP between_IN ] [NP Washington_NNP and_CC Islamabad_NNP ] ._.
```

Figura 4.4: Análisis sintáctico superficial con *OpenNLP*

```
opennlp SentenceDetector en-sent.bin < input_EN.txt | opennlp TokenizerME en-token.bin |
opennlp Parser en-parser-chunking.bin
```

```
(TOP (S (ADVP (RB Early) (PP (IN on) (NP (NNP Saturday)))) (, ,) (NP (
    DT a) (JJ middle-aged) (NNP Pashtun) (NN man)) (VP (VBD used) (NP (
    VBN forged) (NNS documents)) (S (VP (TO to) (VP (VB cross) (PP (IN
    from) (NP (NNP Iran))) (PP (IN into) (NP (NNP Pakistan)))))))) (. .)
    ))
(TOP (S (ADVP (NP (DT A) (JJ few) (NNS hours)) (RB later)) (, ,) (PP (
    IN on) (NP (NP (DT a) (JJ lonely) (NN stretch)) (PP (IN of) (NP (NN
     highway))))) (, ,) (NP (PRP he)) (VP (VBD was) (VP (VBN
    incinerated) (PP (IN by) (NP (DT an) (JJ American) (NN drone))))))
    (. .)))
(TOP (S (NP (PRP It)) (VP (VBZ is) (RB not) (ADVP (RB exactly)) (ADJP (
    JJ clear)) (SBAR (WHADVP (WRB how)) (S (NP (DT the) (NNPS Americans
    )) (VP (VBD tracked) (NP (NP (NNP Mullah) (NNP Akhtar) (NNP
    Muhammad) (NNP Mansour)) (, ,) (NP (NP (NN leader)) (PP (IN of) (NP
     (DT the) (NNP Afghan) (NNP Taliban))) (, ,) (PP (TO to) (NP (NP (
    DT a) (JJ white) (NN sedan)) (VP (VBG rattling) (PP (IN across) (NP
     (NP (DT the) (JJ arid) (NN expanse)) (PP (IN of) (NP (NNP
    Baluchistan) (NNP Province))))))))))))) (. .)))
(TOP (S (S (S (NP (DT The) (NNP United) (NNP States)) (VP (VBD picked)
    (PRT (RP up)) (NP (NP (DT a) (NN mix)) (PP (IN of) (NP (NP (NN
    phone) (NNS intercepts)) (CC and) (NP (NP (NNS tips)) (PP (IN from)
```

```
    (NP (NNS sources))))))))) (, ,) (NP (JJ American) (CC and) (JJ
    European) (NNS officials)) (VP (VBD said))) (, ,) (CC and) (S (NP (
    EX there)) (VP (VBD were) (NP (NNS reports) (SBAR (IN that) (S (NP
    (NNP Pakistan)) (ADVP (RB also)) (VP (VBD provided) (NP (NN
    intelligence)))))))) (. .)))
(TOP (S (S (NP (NNP President) (NNP Obama)) (VP (VBD described) (NP (NP
    (NNP Mullah) (NNP Mansour) (POS 's)) (NN death)) (PP (IN on) (NP (
    NNP Monday))) (PP (IN as) (NP (DT an) (JJ 'important) (NN milestone
    ))))) ('' '') (, ,) (CC but) (S (NP (DT the) (NN strike)) (VP (VBD
    was) (ADVP (RB also)) (NP (NP (DT an) (NN illustration)) (PP (IN of
    ) (NP (NP (DT the) (JJ tangled) (NN relationship)) (PP (IN between)
    (NP (NNP Washington) (CC and) (NNP Islamabad)))))))) (. .)))
```

Figura 4.5: Análisis sintáctico de constituyentes con *OpenNLP*

Ejemplo de procesamiento: texto en español

Se muestra a continuación un ejemplo de procesamiento con *OpenNLP* con un texto en español, en las tareas de reconocimiento de entidades (para la categoría de persona) y de anotación morfosintáctica a partir de un texto periodístico de entrada.

> El presidente del Gobierno en funciones, Mariano Rajoy, ha tenido su primer acto de partido desde que José Manuel Soria presentara su renuncia al cargo este viernes tras el escándalo de los papeles de Panamá y sus empresas offsore en Bahamas y Jersey. Mientras que ayer la vicepresidenta en funciones, Soraya Sáenz de Santamaría, y el ministro de Hacienda en funciones, Cristóbal Montoro, tuvieron que responder a las preguntas de los periodistas en la rueda de prensa posterior a Consejo de Ministros y Soria se haya dedicado a hacer ronda en los medios, Rajoy ha decidido evitar el tema.

Figura 4.6: Texto en español para analizar con *OpenNLP*

```
opennlp SimpleTokenizer < input_opennlp_es.txt |
opennlp T okenNameFinder es-ner-person.bin
```

> El presidente del Gobierno en funciones , <START:person> Mariano Rajoy <END> , ha tenido su primer acto de partido desde que <START:person> José Manuel Soria <END> presentara su renuncia al cargo este viernes tras el escándalo de los papeles de Panamá y sus empresas offsore en Bahamas y Jersey . Mientras que ayer la vicepresidenta en funciones , Soraya Sáenz de Santamaría , y el ministro de Hacienda en funciones , <START:person> Cristóbal Montoro <END> , tuvieron que responder a las preguntas de los periodistas en la rueda de prensa posterior a Consejo de Ministros y Soria se haya dedicado a hacer ronda en los medios , <START:person> Rajoy <END> ha decidido evitar el tema .

Figura 4.7: Reconocimiento de entidades con *OpenNLP*

```
opennlp SimpleTokenizer < input_ES.txt |
opennlp POSTagger opennlp-es-pos-maxent-pos-es.model
```

El_DA presidente_NC del_SP Gobierno_NC en_SP funciones_NC ,_Fc Mariano_AQ Rajoy_NC ,_Fc ha_VAI tenido_VMP su_DP primer_AO acto_NC de_SP partido_NC desde_SP que_PR José_VMI Manuel_AQ Soria_NC presentara_VMS su_DP renuncia_NC al_SP cargo_NC este_DD viernes_NC tras_SP el_DA escándalo_NC de_SP los_DA papeles_NC de_SP Panamá_NC y_CC sus_DP empresas_NC offsore_AQ en_SP Bahamas_NC y_CC Jersey_NC ._Fp Mientras_CS que_CS ayer_RG la_DA vicepresidenta_NC en_SP funciones_NC ,_Fc Soraya_NC Sáenz_AQ de_SP Santamaría_NC ,_Fc y_CC el_DA ministro_NC de_SP Hacienda_NC en_SP funciones_NC ,_Fc Cristóbal_NC Montoro_NC ,_Fc tuvieron_VMI que_CS responder_VMN a_SP las_DA preguntas_NC de_SP los_DA periodistas_NC en_SP la_DA rueda_NC de_SP prensa_NC posterior_AQ a_SP Consejo_NC de_SP Ministros_NC y_CC Soria_NC se_P0 haya_VAS dedicado_VMP a_SP hacer_VMN ronda_NC en_SP los_DA medios_NC ,_Fc Rajoy_NC ha_VAI decidido_VMP evitar_VMN el_DA tema_NC ._Fp

Figura 4.8: Etiquetación morfosintáctica con *OpenNLP*

4.3 GATE

GATE (General Architecture for Text Engineering) es una arquitectura y una plataforma de desarrollo diseñada en la Universidad de Sheffield para la elaboración e integración de aplicaciones en tecnologías del lenguaje. *GATE* posee una distribución *standalone* que permite la ejecución *offline* en tubería de secuencias de módulos de procesamiento lingüístico (*plugins*) para ejecutar todo tipo de tareas de PLN ([Cun+11]). El entorno de procesamiento (*workflow*) de *GATE* se describe más adelante así como su versión *offline* (*GATE Cloud*) y su plataforma colaborativa (*GATE Teamware*), respectivamente, en las secciones 7.1 y 7.5.

Sitio web

`https://gate.ac.uk/`

Versión actual

8.1 (2/6/2015)

Entorno de ejecución

Java 7+

Licencia

GNU Lesser General Public Licence 3.0 (LGPL 3.0)

Herramientas básicas

- Segmentador de frases.
- Tokenizador.
- Reconocedor de entidades nombradas.
- Etiquetador morfosintáctico.
- Analizador sintáctico superficial.
- Analizador sintáctico de constituyentes.
- Analizador sintáctico de dependencias.
- Sistema para la resolución de la correferencia.

Otras herramientas

- Clasificador de documentos: clasifica el texto en una serie de categorías predefinidas.
- Identificación de lengua: determina la lengua de un texto.
- JAPE (Java Annotation Patterns Engine) es un componente de *GATE* basado en expresiones regulares que permite operar sobre los textos anotados lingüísticamente, falicitando el reconocimiento de patrones en las representaciones de tipo grafo usadas en *GATE* y con múltiples aplicaciones en el reconocimiento de entidades o en la extracción de información.

Soporte de idiomas

La única lengua para la que *GATE* ofrece componentes en todas las tareas de procesamiento básico es el inglés. *GATE* distribuye también módulos disponibles para distintas tareas del procesamiento lingüístico en árabe (ar), cebuano (ce), chino (zh), francés (fr), alemán (de), hindí (hi), rumano (ro) y ruso (ru). Véase la Tabla 4.3 para una comparación del soporte de idiomas por tarea.

Tabla 4.3: Soporte de idiomas por herramienta en *GATE*

Herramienta por idioma	ar	ce	zh	fr	en	de	hi	ro	ru
Segmentador de frases					✓		✓		
Tokenizador	✓	✓	✓		✓		✓	✓	
Entidades nombradas	✓	✓	✓	✓	✓	✓		✓	✓
Etiquetador morfosintáctico		✓			✓				✓
Sintaxis superficial					✓				
Sintaxis de constituyentes					✓				
Sintaxis de dependencias					✓				
Resolución de la correferencia					✓				

Formatos de salida

GATE utiliza el modelo TIPSTER para representar las anotaciones (ver sección 3.4.1), y utiliza XML como formato de serialización (ver ejemplo en la tabla 3.4 en la página 27).

Ciclo de vida

GATE distribuye separadamente los módulos de procesamiento (`https://gate.ac.uk/gate/doc/plugins.html`). Cada módulo se distribuye como un fichero Java `.jar` acompañado de un fichero XML de configuración. La creación de un nuevo módulo no supone más que incorporar una nueva clase Java a la librería.

Ejemplo de procesamiento: texto en inglés

Se muestra aquí como ejemplo de procesamiento con *GATE* el resultado, en formato comprimido de una línea (*inline*), de la tarea de reconocimiento de entidades nombradas en un texto periodístico con el plugin ANNIE para el inglés incluido en la distribución estándar de *GATE*.

> Early on Saturday, a middle-aged Pashtun man used forged documents to cross from Iran into Pakistan. A few hours later, on a lonely stretch of highway, he was incinerated by an American drone. It is not exactly clear how the Americans tracked Mullah Akhtar Muhammad Mansour, leader of the Afghan Taliban, to a white sedan rattling across the arid expanse of Baluchistan Province. The United States picked up a mix of phone intercepts and tips from sources, American and European officials said, and there were reports that Pakistan also provided intelligence. President Obama described Mullah Mansour's death on Monday as an 'important milestone', but the strike was also an illustration of the tangled relationship between Washington and Islamabad

Figura 4.9: Texto en inglés para procesar con *GATE*

```
Early on Saturday, a middle-aged Pashtun man used forged documents to cross from <Location
  ↪  gate:gateId="393" locType="country" rule="Location1" ruleFinal="LocFinal">Iran</Location> into
  ↪  <Location gate:gateId="394" rule="Location1" ruleFinal="LocFinal" locType="country"
  ↪  gate:matches="375;380">Pakistan</Location>. A few hours later, on a lonely stretch of highway,
  ↪  he was incinerated by an American drone. It is not exactly clear how the Americans tracked
  ↪  Mullah Akhtar <Person gate:gateId="395" firstName="Muhammad" ruleFinal="PersonFinal"
  ↪  gender="male" surname="Mansour" kind="fullName" rule="PersonFull"
  ↪  gate:matches="376;390">Muhammad Mansour</Person>, leader of the <Organization
  ↪  gate:gateId="396" orgType="other" rule="GazOrganization" ruleFinal="OrgCountryFinal">Afghan
  ↪  Taliban</Organization>, to a white sedan rattling across the arid expanse of <Location
  ↪  gate:gateId="397" rule="LocationPost" kind="locName" ruleFinal="LocFinal">Baluchistan
  ↪  Province</Location>. The <Location gate:gateId="398" locType="country" rule="Location1"
  ↪  ruleFinal="LocFinal">United States</Location> picked up a mix of phone intercepts and tips
  ↪  from sources, American and European officials said, and there were reports that <Location
  ↪  gate:gateId="399" rule="Location1" ruleFinal="LocFinal" locType="country"
  ↪  gate:matches="375;380">Pakistan</Location> also provided intelligence. President <Person
  ↪  gate:gateId="400" rule="GazPerson" ruleFinal="PersonFinal" gender="male" surname="Obama"
  ↪  kind="fullName">Obama</Person> described Mullah <Person gate:gateId="392" rule="Unknown"
  ↪  matchedWithLonger="true" gender="male" gate:matches="376;390" NMRule="Unknown"
  ↪  kind="PN">Mansour</Person>&amp;apos;s death on Monday as an &amp;apos;important milestone&amp;apos;, but
  ↪  the strike was also an illustration of the tangled relationship between <Location
  ↪  gate:gateId="401" locType="city" rule="Location1" ruleFinal="LocFinal">Washington</Location>
  ↪  and <Location gate:gateId="391" locType="city" rule="Location1"
  ↪  ruleFinal="LocFinal">Islamabad</Location>
```

Figura 4.10: Reconocimiento de entidades con *GATE*

4.4 NLTK

NLTK (*Natural Language Toolkit*) es una librería *Python* orientada a la didáctica e investigación del procesamiento del lenguaje natural. Las herramientas de *NLTK* se distribuyen en módulos independientes para cada tarea específica de procesamiento ([BKL09]).

Sitio web

`http://www.nltk.org/`

Versión actual

3.2.1 (4/2016)

Entorno de ejecución

Python 2.7 o 3.2+

Licencia

Apache License 2.0 (APL 2.0)

Herramientas básicas

- Tokenizador: *NLTK* usa de modo predeterminado el Penn Treebank Tokenizer, que utiliza expresiones regulares para tokenizar el texto
- Etiquetador morfosintáctico (*tagger*): el etiquetador morfosintáctico de *NLTK* usa el conjunto de etiquetas del Penn Treebank y está entrenado con este corpus con un modelo de probabilidad de máxima entropía
- Analizador sintáctico superficial (*chunker*): el analizador sintáctico superficial de *NLTK* está entrenado con el corpus ACE (`https://catalog.ldc.upenn.edu/LDC2005T09`) con un modelo de máxima entropía
- Reconocedor de entidades nombradas (*named entity recognizer* o *NER*): el reconocedor de entidades nombradas de *NLTK* está entrenado con el corpus ACE con un modelo de máxima entropía

Soporte de idiomas

NLTK está centrado en el procesamiento del inglés.

Formatos de salida

Los procesadores de *NLTK* poseen un formato propio de salida legible para un ser humano formado por secuencias de cadenas de caracteres.

Ciclo de vida

NLTK distribuye separadamente los módulos de procesamiento y los modelos entrenados. Los modelos son reentrenables. Véase, por ejemplo, la instrucción empleada para entrenar el modelo para el analizador morfosintáctico del inglés a partir del corpus de entrenamiento `conll2000/train.txt`:

```
python train_tagger.py conll2000 --fileids train.txt
```

Ejemplo de procesamiento: texto en inglés

Se muestra a continuación un ejemplo de una cadena de procesamiento en *NLTK*, con la tokenización, anotación morfosintáctica, análisis sintáctico superficial y reconocimiento de las entidades nombradas en la frase «I went to New York to meet John Smith at CUNY University».

```
>>> frase = "I went to New York to meet John Smith at CUNY University";
>>> tokens = nltk.word_tokenize(frase)
>>> tokens
['I', 'went', 'to', 'New', 'York', 'to', 'meet', 'John', 'Smith', 'at', 'CUNY', 'University']
>>> pos_tags = nltk.pos_tag(tokens)
>>> pos_tags
[('I', 'PRP'), ('went', 'VBD'), ('to', 'TO'), ('New', 'NNP'), ('York', 'NNP'), ('to', 'TO'),
 ↪  ('meet', 'VB'), ('John', 'NNP'), ('Smith', 'NNP'), ('at', 'IN'), ('CUNY', 'NNP'),
 ↪  ('University', 'NNP')]
>>> print nltk.ne_chunk(pos_tags)
(S
  I/PRP
  went/VBD
  to/TO
  (GPE New/NNP York/NNP)
  to/TO
  meet/VB
  (PERSON John/NNP Smith/NNP)
  at/IN
  (ORGANIZATION CUNY/NNP University/NNP))
```

Figura 4.11: Ejemplo de procesamiento con *NLTK*

4.5 FreeLing

FreeLing es una biblioteca de C++ que proporciona funcionalidades básicas de procesamiento del lenguaje natural para una amplia variedad de idiomas, incluidas todas las lenguas oficiales en la Península Ibérica (español, gallego, catalán, asturiano, portugués), con excepción del vasco. *FreeLing* también proporciona una interfaz de línea de órdenes que se pueden utilizar para analizar textos y obtener la salida en el formato deseado (XML, JSON, *CoNLL*, *NAF*) ([PS12]).

Sitio web

`http://nlp.lsi.upc.edu/freeling/`

Versión actual

4.0 (2016)

Entorno de ejecución

C++. Siendo *FreeLing* una librería escrita en C++, el uso de C++ como lenguaje de programación de la aplicación proporciona un acceso completo a todas sus funciones. Existen también APIs bastante completas para poder usar *FreeLing* en aplicaciones escritas en los lenguajes de programación *Java*, *Python*, *Perl*, *PHP* y *Ruby*.

Licencia

GNU Affero General Public License (AGPL)

Herramientas básicas

- Segmentador de frases.
- Tokenizador.
- Reconocedor de entidades nombradas.
- Etiquetador morfosintáctico.
- Desambiguador del sentido léxico.
- Analizador sintáctico superficial.
- Analizador sintáctico de constituyentes.
- Analizador sintáctico de dependencias.
- Sistema para la resolución de la correferencia.

Otras herramientas

- Identificación de lengua: determina la lengua de un texto.
- Extracción de grafos semánticos: identifica los eventos descritos en un texto, las relaciones que se establecen entre ellos y los actores que participan.

Soporte de idiomas

FreeLing ofrece distintas herramientas de procesamiento para un conjunto amplio de idiomas, incluidas todas las lenguas oficiales en la Península Ibérica con excepción del vasco. Las lenguas para las que *FreeLing* proporciona algún tipo de procesamiento son el asturiano (as), catalán (ca), alemán (de), inglés (en), francés (fr), gallego (gl), croata (hr), italiano (it), noruego (nb), portugués (pt), ruso (ru), esloveno (sl), español (es) y galés (cy). Véase la Tabla 4.4 para una comparación del soporte de idiomas por tarea.

Tabla 4.4: Soporte de idiomas por herramienta en *FreeLing*

Herramienta por idioma	as	ca	cy	de	en	es	fr	gl	hr	it	nb	pt	ru	sl
Tokenizador	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Segmentador de frases	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Identificación de entidades nombradas	✓	✓	✓		✓	✓	✓	✓		✓		✓	✓	✓
Clasificación de entidades nombradas		✓			✓	✓						✓		
Etiquetador morfosintáctico	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
Desambiguador del sentido léxico		✓			✓	✓	✓	✓	✓					✓
Sintaxis superficial	✓	✓			✓	✓		✓				✓		
Sintaxis de constituyentes	✓	✓			✓	✓		✓				✓		
Sintaxis de dependencias	✓	✓			✓	✓		✓	✓					✓
Resolución de la correferencia					✓	✓								
Extracción de grafos semánticos					✓	✓								

Formatos de salida

FreeLing ofrece diversos tipos de salidas del procesamiento:

- texto: Formato de texto *ad hoc* (legible para un ser humano), con el texto en columnas para los niveles de análisis inferiores y árboles parentetizados para los niveles superiores.
- *CoNLL* (ver sección 3.6.2)
- XML

- JSON
- *NAF* (ver sección 3.5.1)

Ciclo de vida

FreeLing es un desarrollo del TALP Research Center de la Universitat Politècnica de Catalunya. El código fuente en C++ de *FreeLing* se puede descargar de *GitHub*, es altamente modular y está bien documentado, con manuales técnico y de usuario. Los componentes de *FreeLing* pueden modificarse y recompilarse para obtener comportamientos distintos de los codificados, por ejemplo, para producir un formato de salida no previsto por el sistema. El mantenimiento de esta librería es continuo desde el lanzamiento de su primera versión en 2003, con sucesivas mejoras de su rendimiento y ampliación de sus funcionalidades.

Ejemplo de procesamiento: texto en español

Instrucción para la ejecución con *FreeLing* y listado abreviado del resultado en formato XML de la segmentación de frases, tokenización, etiquetado morfosintáctico, lematización, desambiguación semántica, análisis sintáctico de constituyentes, análisis sintáctico de dependencias, resolución de la correferencia del texto y elaboración del grafo semántico del texto «El actor británico Burt Kwouk falleció este martes en Londres a los 85 años. Su viuda pide que recordemos a Burt por sus actuaciones.».

```
analyzer -f es.cfg --lang es --input text --inplv text --loc --outlv semgraph \\
         -s mfs --output xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <wordcount>25</wordcount>
  <cputime>0.489412</cputime>
  <paragraph>
    <sentence id="1">
      <token ctag="DA" form="El" gen="masculine" id="t1.1" lemma="el" num="singular"
 ↪   pos="determiner" tag="DA0MS0" type="article">
        <morpho>
          <analysis ctag="DA" gen="masculine" lemma="el" num="singular" pos="determiner"
 ↪   selected="1" tag="DA0MS0" type="article"/>
        </morpho>
      </token>
      <token ctag="NC" form="actor" gen="masculine" id="t1.2" lemma="actor" num="singular"
 ↪   pos="noun" tag="NCMS000" type="common" wn="09765278-n">
        <morpho>
          <analysis ctag="NC" gen="masculine" lemma="actor" num="singular" pos="noun" selected="1"
 ↪   tag="NCMS000" type="common"/>
        </morpho>
        <senses>
          <sense pgrank="0.0108861" wn="09765278-n"/>
          <sense pgrank="0.00759218" wn="10437852-n"/>
        </senses>
      </token> [...]
      <constituents>
        <node head="1" label="grup-verb">
          <node label="sn">
            <node label="espec-ms">
              <node head="1" label="j-ms">
                <node head="1" leaf="1" token="t1.1" word="El"/></node>
            </node>
            <node head="1" label="grup-nom-ms">
              <node head="1" label="n-ms">
                <node head="1" leaf="1" token="t1.2" word="actor"/></node>
              <node label="s-a-ms">
```

```
                <node head="1" label="a-ms">
                  <node head="1" leaf="1" token="t1.3" word="británico"/></node>
              </node>
            </node>
          </node>
          <node label="sn">
            <node head="1" label="grup-nom-ms">
              <node head="1" label="w-ms">
                <node head="1" leaf="1" token="t1.4" word="Burt_Kwouk"/></node>
            </node>
          </node> [...]
      </constituents>
      <dependencies>
        <depnode function="sentence" token="t1.5" word="falleció">
          <depnode function="suj" token="t1.2" word="actor">
            <depnode function="spec" token="t1.1" word="El"/>
            <depnode function="s.a" token="t1.3" word="británico"/>
            <depnode function="sn" token="t1.4" word="Burt_Kwouk"/></depnode>[...]
      </dependencies>
      <predicates>
        <predicate head_token="t1.2" id="P1.1" sense="play.01" words="actor"/>
        <predicate head_token="t1.5" id="P1.2" sense="die.01|expire.01|perish.01" words="falleció">
          <argument from="t1.1" head_token="t1.2" role="A1" to="t1.4" words="El actor británico
↪   Burt_Kwouk"/>
          <argument from="t1.6" head_token="t1.7" role="AM-TMP" to="t1.7" words="este martes"/>
          <argument from="t1.8" head_token="t1.8" role="AM-LOC" to="t1.9" words="en Londres"/>
          <argument from="t1.10" head_token="t1.10" role="AM-TMP" to="t1.13" words="a los 85 años"/>
        </predicate>
      </predicates>
    </sentence>
    <sentence id="2"> [...]
    </sentence>
  </paragraph>
  <coreferences>
    <coref id="co2">
      <mention from="t1.1" id="m2.1" to="t1.3" words="El actor británico"/>
      <mention from="t1.4" id="m2.2" to="t1.4" words="Burt_Kwouk"/>
      <mention from="t2.7" id="m2.3" to="t2.7" words="Burt"/>
    </coref>
  </coreferences>
  <semantic_graph>
    <entity class="person" id="E1" lemma="burt_kwouk">
      <mention id="t1.2" words="El actor británico"/>
      <mention id="t1.4" words="Burt_Kwouk"/>
      <mention id="t2.7" words="Burt"/>
    </entity>
    <entity id="W3" lemma="[M:??/??/??:??.??:??]">
      <mention id="t1.7" words="este martes"/>
    </entity>
    <entity id="W4" lemma="londres">
      <mention id="t1.9" words="Londres"/>
    </entity> [...]
    <frame id="F2" lemma="die.01|expire.01|perish.01" sense="00358431-v" token="t1.5">
      <argument entity="E1" role="A1:Initial_Location"/>
      <argument entity="W3" role="AM-TMP"/>
      <argument entity="W4" role="AM-LOC"/>
      <argument entity="W5" role="AM-TMP"/> [...]
      <synonym lemma="expirar"/>
      <synonym lemma="fallecer"/>
      <synonym lemma="fenecer"/> [...]
      <URI URI="http://wordnet-rdf.princeton.edu/wn30/00358431-v" knowledgeBase="WordNet"/>
      <URI URI="http://ontologyportal.org/SUMO.owl#Death" knowledgeBase="SUMO"/>
    </frame> [...]
  </semantic_graph>
</document>
```

Figura 4.12: Ejemplo de procesamiento con *FreeLing*

4.6 IXA pipes

La biblioteca *IXA pipes* está formada por un conjunto de utilidades desarrolladas en Java para el procesamiento del lenguaje natural basado en aprendizaje automático y diseñadas para su uso en tubería ([ABR14]).

Sitio web

`http://ixa2.si.ehu.es/ixa-pipes/`

Versión actual

1.1.0 (5/2015)

Entorno de ejecución

La módulos principales necesitan Java 1.7 o superior. *IXA pipes* integra además módulos externos (*third party tools*) que pueden estar escritos en otros lenguajes de programación (como C++ o *Python*).

Licencia

Apache License 2.0 (APL 2.0)

Herramientas básicas

- Segmentador de frases.
- Tokenizador.
- Reconocedor de entidades nombradas.
- Etiquetador morfosintáctico.
- Desambiguador del sentido léxico: *IXA pipes* integra la herramienta externa UKB[1] para la desambiguación léxica.
- Analizador sintáctico superficial.
- Analizador sintáctico de constituyentes.
- Analizador sintáctico de dependencias.
- Sistema para la resolución de la correferencia.

Otras herramientas

- Desambiguador de entidades nombradas y wikificación: determina a que identificador único de la DBpedia se refieren las entidades nombradas identificadas en el texto. *IXA pipes* integra la herramienta externa DBpedia Spotlight para esta tarea.
- Extracción de temas de opinión: identifica las entidades y propiedades que son objeto de opinión en un texto.

Soporte de idiomas

Las *IXA pipes* ofrecen herramientas para el procesamiento en distinto grado del inglés (en), el español (es), el vasco (eu), el gallego (gl), el italiano (it), el neerlandés, el alemán (de) y el francés (fr). Véase la Tabla 4.5 para una comparación del soporte de idiomas por tarea.

[1] `http://ixa2.si.ehu.es/ukb/`

Tabla 4.5: Soporte de idiomas por herramienta en *IXA pipes*

Herramienta por idioma	eu	nl	de	en	fr	gl	it	es
Segmentador de frases	✓	✓	✓	✓	✓	✓	✓	✓
Tokenizador	✓	✓	✓	✓	✓	✓	✓	✓
Lematizador	✓	✓	✓	✓	✓	✓	✓	✓
Etiquetador morfosintáctico	✓	✓	✓	✓	✓	✓	✓	✓
Desambiguador del sentido léxico	✓			✓	✓	✓		✓
Clasificación de entidades nombradas	✓	✓	✓	✓			✓	✓
Sintaxis superficial				✓				
Sintaxis de constituyentes				✓				✓
Sintaxis de dependencias				✓				✓
Resolución de la correferencia				✓				✓
Extracción de temas de opinión				✓				

Formatos de salida

Las *IXA pipes* ofrecen diversos tipos de salidas del procesamiento:

- NAF (ver sección sec:naf). Este es el formato de salida predeterminado de todos los módulos de *IXA pipes*.
- El módulo de tokenización permite la salida del texto tokenizado en una oración por línea y con marcadores para los párrafos, o en formato *CoNLL* (ver sección 3.6.2) con un token por línea, dos saltos de línea para separar las frases y marcadores para los párrafos.
- El analizador sintático de constituyentes permite la salida en el formato necesario para la evaluación con el programa *Evalb*[2].
- El módulo de clasificación de entidades nombradas permite la salida en los formatos *CoNLL03* y *CoNLL02*, y en el formato nativo de *OpenNLP*.
- El módulo de extracción de temas de opinión permite la salida en el formato nativo de *OpenNLP*.

Ciclo de vida

IXA pipes es un desarrollo del Grupo IXA de la Universidad del País Vasco. El código fuente en Java se puede descargar de *GitHub*, es altamente modular y está bien documentado. Los modelos se distribuyen separadamente del código. *IXA pipes* permite reentrenar fácilmente las herramientas incluidas en la librería especificando los parámetros del entrenamiento (por ejemplo, la localización de los modelos para el entrenamiento) en un fichero. Véase, por ejemplo, la instrucción para entrenar el módulo de reconocimento de entidades nombradas a partir de las especificaciones contenidas en el fichero `trainParams.properties`:

```
java -jar ixa-pipe-pos-1.5.1/target/ixa-pipe-pos-1.5.1.jar train \\
     -p trainParams.properties
```

[2] `http://nlp.cs.nyu.edu/evalb/`

Ejemplo de procesamiento: texto en español

Instrucción para la ejecución con *IXA pipes* y listado abreviado del resultado en formato NAF de la segmentación de frases, tokenización, etiquetado morfosintáctico, lematización, clasificación de entidades nombradas y análisis sintáctico de constituyentes del texto «El fundador de Amazon, Jeff Bezos, ha alzado la voz contra Peter Thiel y en defensa del portal estadounidense de cotilleos Gawker. Ante el público de Los Ángeles, Bezos también señaló que la libertad de expresión está garantizada.».

```
java -jar ixa-pipe-tok-1.8.4/target/ixa-pipe-tok-1.8.4.jar tok -l es |
java -jar ixa-pipe-pos-1.5.1/target/ixa-pipe-pos-1.5.1.jar tag \\
     -m morph-models-1.5.0/es/es-pos-perceptron-autodict01-ancora-2.0.bin\\
     -lm morph-models-1.5.0/es/es-lemma-perceptron-ancora-2.0.bin |
java -jar  -Xmx1g ixa-pipe-nerc-1.6.0/target/ixa-pipe-nerc-1.6.0-exec.jar tag\\
     -m nerc-models-1.5.4/es/es-4-class-clusters-ancora.bin |
java -jar ixa-pipe-parse-1.1.2/target/ixa-pipe-parse-1.1.2.jar parse \\
     -m parse-models/es-parser-chunking.bin
```

```
<?xml version="1.0" encoding="UTF-8"?>
<NAF xml:lang="es" version="v1.naf">
  <nafHeader>
    <linguisticProcessors layer="text">
      <lp name="ixa-pipe-tok-es" [...] />
    </linguisticProcessors>
    <linguisticProcessors layer="terms">
      <lp name="ixa-pipe-pos-es-pos-perceptron-autodict01-ancora-2.0" [...] />
    </linguisticProcessors>
    <linguisticProcessors layer="entities">
      <lp name="ixa-pipe-nerc-es-4-class-clusters-ancora" [...] />
    </linguisticProcessors>
    <linguisticProcessors layer="constituency">
      <lp name="ixa-pipe-parse-es-parser-chunking" [...] />
    </linguisticProcessors>
  </nafHeader>
   <text>
    <wf id="w1" offset="0" length="2" sent="1" para="1">El</wf>
    <wf id="w2" offset="3" length="8" sent="1" para="1">fundador</wf>
    <wf id="w3" offset="12" length="2" sent="1" para="1">de</wf>
    <wf id="w4" offset="15" length="6" sent="1" para="1">Amazon</wf> [...]
  </text>
    <terms>
    <!--El-->
    <term id="t1" type="close" lemma="el" pos="D" morphofeat="DA0MS0">
      <span>
        <target id="w1" />
      </span>
    </term>
    <!--fundador-->
    <term id="t2" type="open" lemma="fundador" pos="N" morphofeat="NCMS000">
      <span>
        <target id="w2" />
      </span>
    </term>
    <!--de-->
    <term id="t3" type="close" lemma="de" pos="P" morphofeat="SPS00">
      <span>
        <target id="w3" />
      </span>
    </term>
    <!--Amazon-->
    <term id="t4" type="close" lemma="amazon" pos="R" morphofeat="NP00000">
      <span>
        <target id="w4" />
      </span>
```

```
    </term> [...]
  </terms>
    <entities>
    <entity id="e1" type="ORG">
      <references>
        <!--Amazon-->
        <span>
          <target id="t4" />
        </span>
      </references>
    </entity>
    <entity id="e2" type="PER">
      <references>
        <!--Jeff Bezos-->
        <span>
          <target id="t6" />
          <target id="t7" />
        </span>
      </references>
    </entity> [...]
  </entities>
    <constituency>
    <tree>
      <!--Non-terminals-->
      <nt id="nter1" label="TOP" />
      <nt id="nter2" label="SENTENCE" />
      <nt id="nter3" label="SN" />
      <nt id="nter4" label="DAOMSO" /> [...]
      <!--Terminals-->
      <!--El-->
      <t id="ter1">
        <span>
          <target id="t1" />
        </span>
      </t>
      <!--fundador-->
      <t id="ter2">
        <span>
          <target id="t2" />
        </span>
      </t> [...]
      <!--Tree edges-->
      <edge id="tre2" from="nter2" to="nter1" />
      <edge id="tre3" from="nter3" to="nter2" />
      <edge id="tre4" from="nter4" to="nter3" /> [...]
    </tree>
    <tree> [...]
    </tree>
  </constituency>
</NAF>
```

Figura 4.13: Ejemplo de procesamiento con *IXA pipes*

4.7 Resumen comparativo por idioma

A continuación se ofrece, en forma de tablas, un resumen comparativo de las funcionalidades de cada librería por idioma, teniendo en cuenta en la comparación el inglés, el español y las lenguas cooficiales.

Tabla 4.6: Soporte de inglés por herramienta

Soporte de inglés	CoreNLP	OpenNLP	GATE	NLTK	FreeLing	IXA pipes
Segmentación de frases	✓	✓	✓		✓	✓
Tokenización	✓	✓	✓	✓	✓	✓
Etiquetación morfosintáctica	✓	✓	✓	✓	✓	✓
Lematización	✓				✓	✓
Desambiguación léxica					✓	✓
Identificación de entidades	✓	✓	✓	✓	✓	✓
Clasificación de entidades	✓	✓	✓	✓	✓	✓
Sintaxis superficial		✓	✓	✓	✓	✓
Sintaxis de constituyentes	✓	✓	✓		✓	✓
Sintaxis de dependencias	✓		✓		✓	✓
Correferencia	✓	✓	✓		✓	✓

Tabla 4.7: Soporte de español por herramienta

Soporte de español	CoreNLP	OpenNLP	GATE	NLTK	FreeLing	IXA pipes
Segmentación de frases	✓				✓	✓
Tokenización	✓				✓	✓
Etiquetación morfosintáctica	✓	✓			✓	✓
Lematización					✓	✓
Desambiguación léxica					✓	✓
Identificación de entidades	✓	✓			✓	✓
Clasificación de entidades	✓	✓			✓	✓
Sintaxis superficial					✓	
Sintaxis de constituyentes	✓				✓	✓
Sintaxis de dependencias					✓	✓
Correferencia					✓	✓

Tabla 4.8: Soporte de gallego por herramienta

Soporte de gallego	CoreNLP	OpenNLP	GATE	NLTK	FreeLing	IXA pipes
Segmentación de frases					✓	✓
Tokenización					✓	✓
Etiquetación morfosintáctica					✓	✓
Lematización					✓	✓
Desambiguación léxica					✓	✓
Identificación de entidades					✓	
Clasificación de entidades						
Sintaxis superficial					✓	
Sintaxis de constituyentes					✓	
Sintaxis de dependencias					✓	
Correferencia						

Tabla 4.9: Soporte de catalán por herramienta

Soporte de catalán	CoreNLP	OpenNLP	GATE	NLTK	FreeLing	IXA pipes
Segmentación de frases					✓	
Tokenización					✓	
Etiquetación morfosintáctica					✓	
Lematización					✓	
Desambiguación léxica					✓	
Identificación de entidades					✓	
Clasificación de entidades					✓	
Sintaxis superficial					✓	
Sintaxis de constituyentes					✓	
Sintaxis de dependencias					✓	
Correferencia						

Tabla 4.10: Soporte de vasco por herramienta

Soporte de vasco	CoreNLP	OpenNLP	GATE	NLTK	FreeLing	IXA pipes
Segmentación de frases						✓
Tokenización						✓
Etiquetación morfosintáctica						✓
Lematización						✓
Desambiguación léxica						✓
Identificación de entidades						✓
Clasificación de entidades						✓
Sintaxis superficial						
Sintaxis de constituyentes						
Sintaxis de dependencias						
Correferencia						

5. Entornos de procesamiento

Siguiendo a [EG14] diremos que un **componente** (también llamadao **procesador**) es una **herramienta** integrada dentro de un **entorno de procesamiento** (*processing framework*) mediante la implementación un **adaptador** entre la herramienta y el entorno. Así, el entorno de procesamiento define una interfaz de programación uniforme, un modelo de datos y un modelo de procesamiento que permiten la interoperabilidad estructural entre componentes. Muchos componentes son genéricos y requieren algún recurso específico, como por ejemplo un lexicón o un modelo probabilista. Se denominará **selección de recursos** a la elección de recursos y la configuración de un componente para que los use, y denominaremos **adquisición de recursos** a la tarea de obtención de los recursos. Según [EG14], los recursos deberían especificarse mediante varias coordenadas: herramienta, lengua, variante y versión.

Los entornos de procesamiento ofrecen acceso a componentes de PLN pero los usuarios pueden añadir nuevos. Los entornos también ofrecen maneras de describir **flujos de trabajo** o **cadenas de procesamiento** (*pipelines*) para configurar y ejecutar una cadena de componentes de PLN sobre documentos proporcionados por un usuario. Los usuarios también debe poder editar o configurar los formatos de salida de los componentes pero no tienen la posibilidad de introducir nuevos formatos en el entorno de procesamiento.

Plataformas como *UIMA* o *GATE*, las dos plataformas más populares en la actualidad, utilizan modelos de anotación basados en referencias y estructuras de rasgos, descritos en *UIMA Type System* o *GATE Annotation Schema*. Ambos describen sus cadenas de procesamiento mediante documentos XML que hacen referencia a componentes individuales por su nombre, pero no por su versión, y en los que no se incluye el aprovisionamiento del componente o de los recursos a usar por el componente, que podrían estar o en algún repositorio externo al marco de procesamiento. *UIMA* y *GATE* se centran más en un tipo de estructuras para anotación que en integrar servicios distribuidos y fuentes heterogéneas. Existen otros proyectos para la construcción de componentes de propósito general basados en *UIMA* como *DKPro* o *ClearTK* [OWB09] con sistemas de tipos que describen una gran

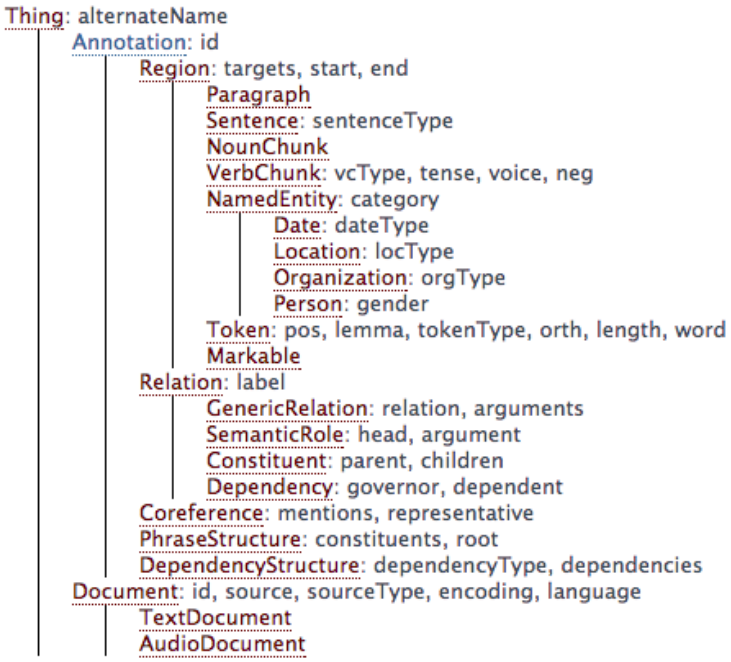

Figura 5.1: Jerarquía de tipos del vocabulario de intercambio de componentes en *LAPPS*.

variedad de fenómenos lingüísticos. Algunos sistemas como *LAPPS* definen una ontología[1] para un núcleo de objetos y rasgos lingüísticos intercambiados ente componentes de PLN que consumen y producen datos anotados lingüísticamente [Ide+14]. Esta jerarquía de tipos, puede verse en la figura 5.1, pretende ser ser utilizada para la descripción de la entrada y salida de componentes, y dar soporte al descubrimiento de servicios de PLN, para que puedan componerse y reutilizarse.

Los **sistemas de gestión de flujos de trabajo** permiten que tanto usuarios expertos como usuarios sin conocimientos en PLN puedan llevar a cabo tareas complejas de PLN sobre colecciones de documentos sin necesidad de usar la línea de comandos y sin tener que conocer los formatos de intercambio entre herramientas. Este tipo de sistemas permiten acceder a herramientas ya instaladas de PLN, haciendo innecesaria su instalación local. Un editor de flujos de trabajo debe satisfacer las demandas de dos tipos de usuarios extremos: usuarios sin ningún tipo de conocimiento de programación y usuarios expertos. El editor debería tener un buen soporte gráfico y permitir operaciones de *drag & drop* e incluir varios tipos de asistentes. Los flujos así definidos deberán poderse guardar en formato estándar de intercambio, como documento XML o JSON. El editor de flujos de trabajo debe permitir buscar y hojear el registro de flujos de trabajo predefinidos.

Los entornos de procesamiento promueven y favorecen la sostenibilidad, manejabilidad, usabilidad e interoperabilidad de componentes de PLN. Estos entornos deben, idealmente, satisfacer algunos requirimientos: Por una lado, deben se accesibles a una amplio rango de usuarios, desde expertos desarrolladores de PLN hasta los no expertos en PLN; los análisis deben ser reproducibles; los usuarios pueden compartir y publicar análisis a través de la web y de manera interactiva. Estos entornos deben están bien desarrollandos y

[1] `http://vocab.lappsgrid.org/`

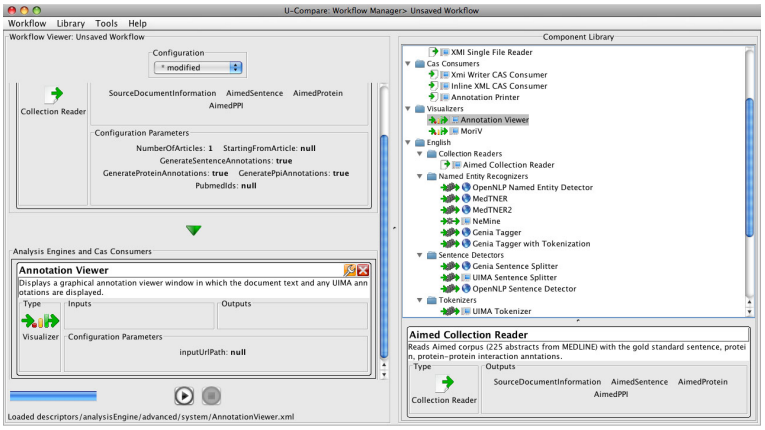

Figura 5.2: Editor de flujo de trabajos en *U-Compare*.

mantenidos, y sería deseable que fueran sistemas de código abierto. También es interesante que estos entornos tengan distintas opciones para ser ejecutados en escenarios distintos con distintos requerimientos de privacidad, volumen de datos o potencia de cálculo: a través de Internet; localmente mediante una descarga de una instancia por clonación del proyecto, enpaquetamiento en máquinas virtuales o contenedores; o en una la nube. Hay muchos modelos basados en distintas tecnologías, a continuación se introducen algunos de los entornos de procesamiento más populares en PLN.

5.1 Apache uimaFIT

Los componentes *UIMA* se configuran mediante ficheros descriptores XML que indican al *framework* cómo deben instanciarse y desplegarse en tiempo de ejecución [Roe+09]. *UimaFIT*[2] proporciona un mecanismo de anotaciones Java para describir directamente en el código fuente de los componentes *UIMA*. En general, uimaFIT aborda la simplificación de la implementación de componentes y su instanciación.

5.2 U-Compare

U-Compare [Kan+11] es una aplicación Java que permite, mediante una interfaz gráfica de usuario, crear y ejecutar cadenas de procesamiento *UIMA* para comparar sus resultados. Algunos de los componentes de la cadena son locales y otros llaman a servicios web en distintas instalaciones remotas. Las cadenas de procesamiento son creadas mediante una interfaz gráfica de usuario. En las figuras 5.2, 5.3 y 5.4 muestran, respectivamente, el editor de flujo de trabajos, un visor de anotaciones y otro de resultados de U-Compare.

5.3 Kachako

Kachako[3] es una evolución de U-Compare que pretende solventar los problemas de escalabilidad y automatización de tareas [Kan12]. *Kachaco* parece estar todavía en

[2] `http://uima.apache.org/uimafit.html`
[3] `http://kachako.org/`

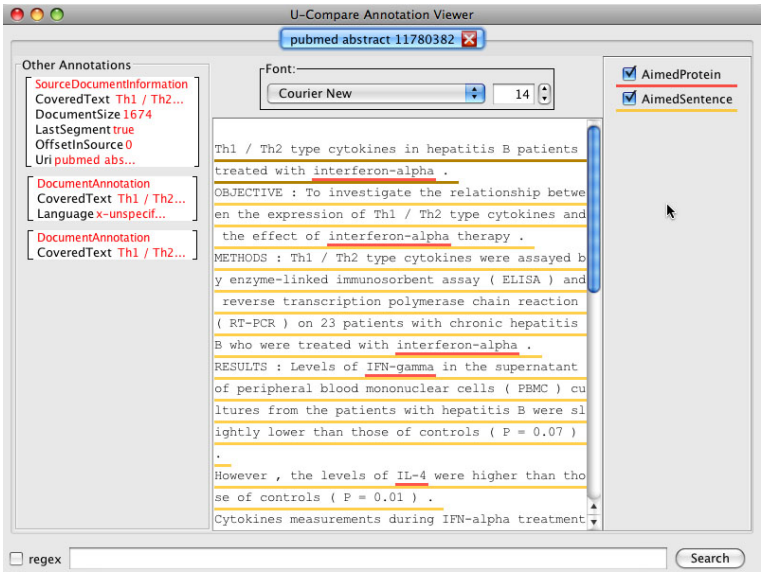

Figura 5.3: Visor de anotaciones en *U-Compare*: Proteinas en resúmenes de PubMed.

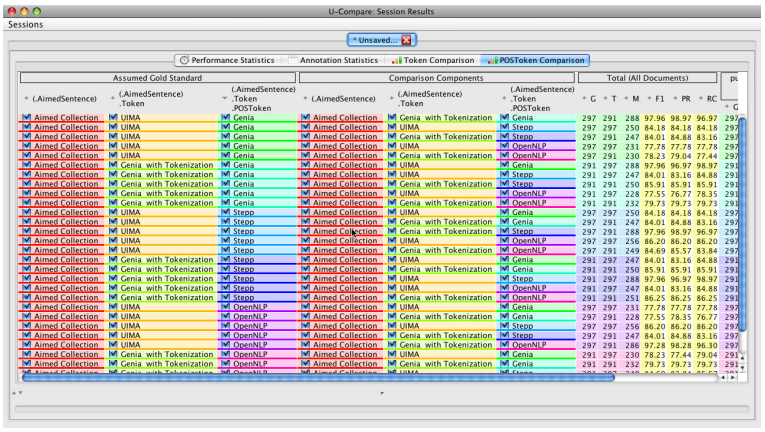

Figura 5.4: Resultados de una session en *U-Compare*.

desarrollo o versión alfa.

5.4 Argo

Argo[4] es un *workbench* basado en *UIMA* con un interfaz web de usuario para la creación de anotaciones, automáticas o manuales, derivadas de componentes de minería de textos [Rak+13]. *Argo* está inspirado en *U-Compare* y sus características más importantes y diferenciadoras son:

- hay un acceso completo a través de navegadores web mediante llamadas asíncronas.
- el procesamiento definido en un flujo de trabajo es llevado a cabo remotamente.
- incorpora componentes de procesamiento interactivos diseñados para no expertos.
- los usuarios pueden compartir flujos de trabajo, datos y resultados con otros.
- permite a los desarrolladores que construyan clientes propios que se comuniquen con los servicios web ofrecidos.

Argo está constituido por una serie de objetos organizados en cuatro categorías:

- documentos: recursos que contienen texto y son objeto de procesamiento. Varios documentos pueden agruparse en directorios y subdirectorios.
- componentes: objetos de procesamiento que esperan una entrada *CAS* que contenga anotaciones de un determinado tipo.
- flujos de trabajo: múltiples componentes de procesamiento interconectados en un orden específico definido por un usario. En la figura 5.5 puede verse el editor de flujos de trabajo de Argo.
- ejecuciones: un listado de las ejecuciones actuales y pasadas de flujos de trabajo con información como el tiempo de ejecución, su duración y progreso actual.

El sistema viene con un conjunto predeterminado de componentes y flujos de trabajo para varias tareas, entre otras la segmentación de oraciones o palabras, el reconocimiento de entidades nombradas, o el almacenamiento en bases de datos. Cualquier usuario puede depositar sus componentes si cumplen con la especificación *UIMA*. Un tipo de componente especial es el componente interactivo que requiere la intervención del usuario, en la mayoría de los casos, para introducir alguna anotación manualmente. Argo proporciona un editor de anotaciones de propósito general que permite añadir nuevas anotaciones, eliminar o modificar anotaciones existentes y añadir metadatos.

La comunicación entre el cliente y el servidor se realiza por medio de protocolos para servicios web, *SOAP* y *REST*. El servidor es un balanceador de carga que realiza las peticiones a otros servidores dedicados.

Por ser *Argo* un sistema orientado hacia el procesamiento de literatura biomédica, algunos componentes permiten realizar búsquedas en repositorios como *MEDLINE*[5] para recuperar resúmenes de artículos. Argo permite definir también algunos componentes por agregación de otros en una cadena de procesamiento.

[4] http://argo.nactem.ac.uk/

[5] `https://www.nlm.nih.gov/bsd/pmresources.html`

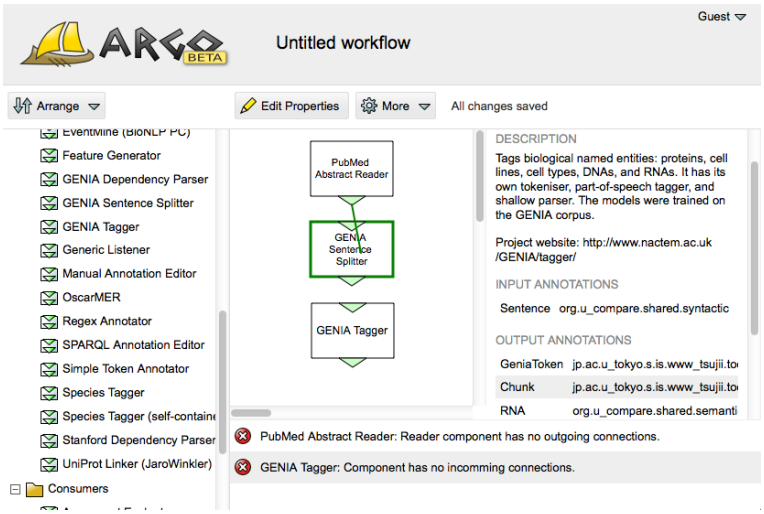

Figura 5.5: Editor de flujo de trabajos en *Argo*.

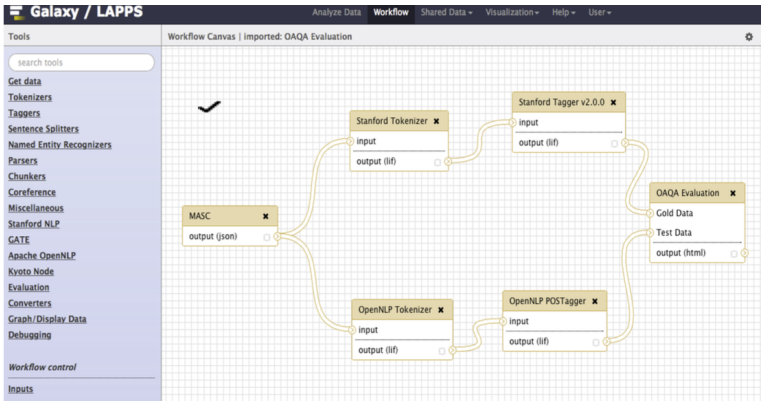

Figura 5.6: Editor de flujo de trabajos para comparación de resultados en *Galaxy/LAPPS*.

5.5 Galaxy

Galaxy[6] es un sistema de código abierto para la gestión a través de web de flujos de trabajo. *Galaxy* fué desarrollado inicialmente para investigación con datos masivos en genómica y bioinformática. Los portales basados en *Galaxy* permiten que biólogos sin conocimientos de programación accedan y configuren herramientas de procesamiento, lleven a cabo experimentos y compartan sus resultados.

El *Language Analysis Portal*[7] (LAP), desarrollado en la Univeridad de Oslo dentro del proyecto CLARIN, utiliza *Galaxy* para la definición de flujos de tareas [Lap+13]. Permite integrar diferentes tipos de servicios así como componentes de *UIMA* o *GATE*. También *LAPPS*[8] (Language Application Grid) usa *Galaxy* para crear cadenas de procesamiento con herramientas de *GATE*, *Apache OpenNLP* (véase sección 4.1) o *Stanford CoreNLP* (sección 4.1).

[6] `https://galaxyproject.org`
[7] `http://www.mn.uio.no/ifi/english/research/projects/clarino/`
[8] `http://galaxy.lappsgrid.org`

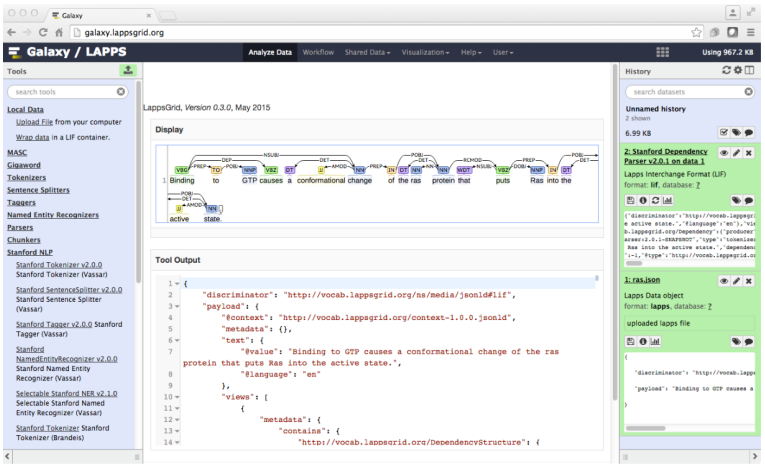

Figura 5.7: Visor de análisis de dependencias en *Galaxy/LAPPS*.

5.6 Taverna

Taverna[9] [Hul+06; Oin+07] es una plataforma o sistema para la composición y gestión de servicios Web dentro del proyecto MyGrid. *Taverna* hace posible llevar a cabo experimentaciones *in silico* usando flujos de trabajo. Estos flujos de trabajo en *Taverna* son representaciones abstractas de los pasos y las interacciones en el proceso de recolección y procesamiento de datos biológicos almacenados en forma digital. *Taverna* tiene varias capas de procesamiento: (i) una capa de flujo de datos permite a los científicos representan un flujos de trabajo desde una perspectiva orientada a los problemas (lo que se produce y lo que se consume en cada servicio con el fin de obtener un resultado final), (ii) una capa de ejecución que se encarga de ejecutar el flujo de trabajo y de ocuparse de las iteraciones y la transformación de los datos, y (iii) la ejecución/invocación real de los servicios web que participan en el flujo de trabajo. Toda la interacción se gestiona a través de una interfaz gráfica de usuario.

Además de estas tres capas, también se proporcionan mecanismos para la adición de funcionalidades, añadir nuevos procesadores, asi como proporcionar instantáneas de la ejecución a través de mensajes.

Taverna puede ser configurado para tolerar fallos (por ejemplo, la reejecución de los procesos después del fracaso o llamar a los servicios equivalentes cuando uno falla), y también para descubrir nuevos servicios a través de una serie de mecanismos de exploración de repositorios públicos de servicios web indexados (aprovechando los ficheros de descripción de servicios web) o permitir a los usuarios registrar o integrar sus propios servicios. Una de las limitaciones de Taverna es su capacidad para informar a los usuarios sobre estado de ejecución o fallos, debido a su dependencia de los servicios web. En la figura 5.8 se muestra un flujo de trabajo donde se usa *U-Compare*.

[9] `https://taverna.incubator.apache.org/`

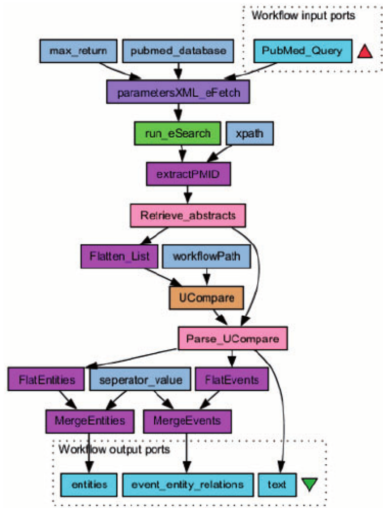

Figura 5.8: Flujo de trabajo en *Taverna* que contiene una actividad con *U-Compare*.

5.7 Heart of Gold

Heart of Gold[10] es un *middleware* para la integración de componentes de PLN de análisis profundo y superficial [Sch05]. Este entorno ofrece una infraestructura flexible para la construcción de aplicaciones que usen RMRS (*Robust Minimal Recursion Semantics*) o anotaciones *stand-off* XML.

5.8 Vistrails

Vistrails[11] es un sistema de código abierto para la gestión de flujos de procesamiento científico con soporte a la exploración y visualización de datos. El sistema está diseñado para admitir las modificaciones a las cadenas de procesamiento típicas de entornos de experimentación y simulación en los que se generan y evaluan hipótesis sobre un conjunto de datos a estudio.

5.9 Kepler

Kepler[12] es otro sistema de gestión de cadenas de procesamiento en entornos científicos multidisciplinares con posibilidad de compartición de modelos y análisis. *Kepler* opera sobre datos almacenados localmente o en internet en diversos formatos. El entorno está diseñado para la integración de componentes muy heterogéneos o para facilitar la ejecución remota o distribuida de modelos. Tanto las fuentes como los componentes de una cadena de procesamiento pueden ser seleccionados y conectados a través de una interfaz gráfica. Todos los componentes, los datos y las cadenas de procesamiento pueden ser compartidos a través del sistema. Hay algunos ejemplos de aplicación de Kepler a tareas de PLN, como [Bri+12] o [Goy+16].

[10] `http://heartofgold.dfki.de`
[11] `http://www.vistrails.org`
[12] `https://kepler-project.org/`

```
#!/usr/bin/env groovy
...
runPipeline(
  createReaderDescription(TextReader,
    TextReader.PARAM_SOURCE_LOCATION, "document.txt",
    TextReader.PARAM_LANGUAGE, "en"),
  createEngineDescription(OpenNlpSegmenter),
  createEngineDescription(OpenNlpPosTagger),
  createEngineDescription(LanguageToolLemmatizer),
  createEngineDescription(MaltParser),
  createEngineDescription(Conll2006Writer,
    Conll2006Writer.PARAM_TARGET_LOCATION, "."));
```

Figura 5.9: *Script* en *Groovy* que produce un análisis de dependencias en formato CoNLL en *DKPro Core*.

5.10 ALPE

ALPE (*Automatic Linguistic Processing Environment*) es un sistema diseñado para facilitar el despliegue y manejo de grandes colecciones dinámicas de recursos y herramientas lingüísticas [CP08]. *ALPE* puede construir cadenas de procesaminto lingüístico teniendo en cuenta los formatos de anotación y herramientas integrados en una estructura jerárquica que fué desarrollada para el proyecto LT4eL[13]. El objetivo principal de *ALPE* es facilitar al usuario no experto en programación la interacción con el sistema, permitiéndole integrar tanto herramientas como recursos, y procesar documentos con cadenas que se infieren automáticamente a partir del formato del documento de entrada y del formato de salida. En caso de haber varias cadenas, estas son presentadas al usuario para que elija.

5.11 TextGrid

TextGrid[14] pretende ser un entorno virtual para la investigación en humanidades: filología, lingüística, musicología o historia del arte [NLS11]. Este entorno ofrece varias herramientas y servicios para la creación, análisis, edición y publicación de texto e imágenes. TexGrid está constituido por un laboratorio y un repositorio que ofrece almacenamiento, preservación y compartición de datos científicos. El laboratorio de *TexGrid* es un software de código abierto que pueder ejecutarse desde cualquier ordenador y permite acceder a herramientas especiales, servicios y contenido.

5.12 WebLicht

WebLicht[15] (*Web-based Linguistic Chaining Tool*) es una infraestructura distribuida de servicios web de PLN hospedadas en distintas instalaciones [HZH10]. Los componentes

[13] http://www.lt4el.eu/
[14] https://textgrid.de/en/
[15] http://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/Main_Page

```
<topology>
  <cluster componentsBaseDir="/home/worker/components"/>
  <module name="EHU-tok" runPath="EHU-tok.v21/run.sh"
          input="raw" output="text" procTime="1"/>
  <module name="EHU-pos" runPath="EHU-pos.v21/run.sh"
          input="text" output="terms"
          procTime="2" source="EHU-tok"/>
  <module name="EHU-nerc" runPath="EHU-nerc.v21/run.sh"
          input="terms" output="entities"
          procTime="11" source="EHU-pos"/>
  <!-- ... -->
</topology>
```

Figura 5.10: Descripción de un pipeline en NewsReader

intercambian su información en un formato XML llamado TCF (*Text Corpus Format*, [Hei+10] o ver §3.5.7). Las cadenas de procesamiento son creadas y ejecutadas usando la interfaz web de *WebLicht*. Cuando por alguna razón algunas herramientas no son combinables, *WebLicht* no permite combinar dichas herramientas, p. ej. un analizador sintáctico con un analizador morfosintáctico cuyas etiquetas no son compatibles. Muchos niveles de anotación lingüística tienen herramientas de visualización compatibles con TCF.

5.13 DKPro Core

DKPro se define como una comunidad de proyectos sobre PLN de tipología amplia, como el procesamiento lingüístico, aprendizaje automático o recursos léxicos. *DKPro Core*[16] es una colección de componentes para PLN basados en *UIMA*. Las cadenas de procesamiento de *DKPro Core* están implementadas en *Groovy*[17], un lenguaje de *scripting* basado en Java. Puede verse un ejemplo de *script* que produce un análisis de dependencias en formato CoNLL en la figura 5.9.

5.14 Newsreader

Dentro del proyecto *NewsReader*[18] se desarrolló un sistema complejo de extracción de eventos en varios idiomas. El sistema combina varios procesadores, entre los que cabe destacar las *IXA pipes* (véase sección 4.6) para el procesamiento básico, y herramientas como UKB o MATE para la desambiguación de acepciones y SRL, respectivamente. Estos módulos son conectados en forma de *pipeline*, y la integración se produce porque todos ellos consumen y producen documentos utilizando el formato NAF (véase sección 3.5.1).

NewsReader también diseñó un sistema de descripción de flujos de trabajos, de tal forma que la combinación de los diferentes módulos puede describirse de forma declarativa por medio de un fichero XML, tal como se muestra en la figura 5.10.

[16] `https://dkpro.github.io`
[17] `http://groovy.codehaus.org`
[18] `http://www.newsreader-project.eu`

6. Esquemas de ejecución y distribución

En los capítulos anteriores se presentaron las cadenas de procesamiento lingüístico y sus entornos de procesamiento actuales. En este capítulo se describen los llamados esquemas de ejecución y distribución, que permiten desplegar una cadena PLN en entornos distribuidos, y ejecutarla eficientemente. El capítulo cubre diversos aspectos referentes tanto a los diferentes esquemas de ejecución (por lotes o en línea), la distribución de sistemas (computación distribuida) así como el empaquetado, configuración y orquestación de servicios o la computación en la nube.

Es difícil hacer una comparación genérica sobre los diferentes esquemas y sus implementaciones (en términos de velocidad, uso de CPU, memoria, red). Normalmente los costes y la velocidad de procesado son dos dimensiones a balancear. Uno de los aspectos más inmediatos a escoger es la cantidad de las máquinas a utilizar, y la potencia de las mismas. Esto usualmente depende de nuestro tipo de tareas. Así, escoger entre muchas máquinas pequeñas (mas baratas) o menos máquinas pero mas potentes o una sola máquina con grandes recursos, el numero de nodos a utilizar, etc. dependerá en gran medida de nuestros requerimientos técnicos y presupuesto. Normalmente la velocidad de procesado se incrementa con el incremento del número de nodos que utilicemos hasta llegar a un punto de saturación.

En el ámbito académico existen, aún, pocas comparaciones especificas para tareas de procesamiento del lenguaje natural sobre la eficiencia de los diferentes esquemas y su escalabilidad. Uno de los pocos ejemplos para tareas de PLN es [Cha+11] donde se muestra en el domino médico para una herramienta concreta como se aumenta el *throughput* del sistema (la velocidad de procesado de los documentos) a costa de aumentar el precio del procesado por documento (más CPUs, más memoria). Incluso encontrar la mejor configuración para una tarea especifica puede resultar no trivial, por ejemplo si queremos etiquetar entidades nombradas nuestros críticas (*reviews*) usando *CoreNLP* de Stanford en

un *cluster* de Amazon[1].

6.1 Modos de procesamiento

Los sistemas de ejecución de las aplicaciones se pueden dividir en dos grandes grupos, por lotes (*batch*) o en línea (*streaming*). En la ejecución por lotes (o tareas) toda la información a procesar es conocida en el momento de empezar la ejecución de la tarea, mientras que la ejecución en línea (o en tiempo real) se procesa la información de manera continua a medida que esta entra en el sistema.

Los modelos de procesamiento en línea se basan en servicios de colas, también denominados servicio de mensajes (*message brokers*). Uno de los primeros estándares al respecto es el servicio de mensajería instantánea, también conocido como *Middleware Orientado a Mensajes (MOM)*. Por ejemplo, *Java Message Service* (JMS) es una librería API *MOM* para aplicaciones Java que implementa dos protocolos de comunicación: el modelo punto a punto y el publicación/suscripción. Ambos modelos pueden ser síncronos (*receive*) o asíncronos (*MessageListener*).

También hay que mencionar el estándar *Advanced Message Queuing Protocol (AMQP)*[2], un protocolo de estándar abierto en la capa de aplicaciones de un sistema de comunicación. Las características que definen al protocolo *AMQP* son la orientación a mensajes, encolamiento, enrutamiento (tanto punto-a-punto como publicación-subscripción), exactitud y seguridad. *AMQP* estipula el comportamiento tanto del servidor que provee los mensajes como del cliente de la mensajería para que las implementaciones de diferentes proveedores sean interoperables, de la misma manera que los protocolos SMTP, HTTP, FTP.

En la actualidad hay una gran diversidad de sistemas de colas disponibles. Los principales sistemas en el mercado actual son: *Kafka*[3], *ActiveMQ*[4], *Kestrel*[5], *RabbitMQ*[6], *Amazon Kinesis Streams*[7]. Estas implementaciones (algunas veces intercambiables) son la base de los diferentes sistemas y arquitecturas que se presentarán en los siguientes apartados[8].

6.1.1 Sistemas de ejecución especificos de PLN

Las características específicas de la herramientas de PLN hizo que en las primeras etapas predominasen las arquitecturas centralizadas con sistemas de ejecución por lotes. Los siguientes apartados presentan los dos grandes sistemas tradicionalmente empleados en PLN, *GATE* y *UIMA*.

GATE

La función de la arquitectura *GATE* (ver sección 4.3) es la integración de técnicas de procesamiento lingüístico que pueden llevarse a cabo en ella. La integración permite la construcción de sistemas destinados a una tarea determinada, a partir de una serie de módulos que realizan un proceso lingüístico especifico. Estas tareas pueden pertenecer a

[1] ver `http://tech.reputation.com/natural-language-processing-with-apache-spark/`
[2] `https://es.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol`
[3] `http://kafka.apache.org`
[4] `http://activemq.apache.org/`
[5] `https://github.com/twitter-archive/kestrel/blob/master/docs/guide.md`
[6] `https://www.rabbitmq.com`
[7] `https://aws.amazon.com/es/kinesis/streams/`
[8] La comparación entre las diferentes sistemas de colas queda fuera del alcance de este documento.

niveles diversos del PLN, desde anotadores y analizadores sintácticos hasta sistemas de extracción de información.

GATE se basa en una arquitectura centraliza de ejecución por lotes con su propia solución de escalabilidad (*GateCloud*) que se presenta en el siguiente capitulo (ver sección 7.1).

GATE distingue diferentes tipos de componentes:

- *Language resources*: recursos tales como diccionarios, ontologías, etc. necesarios para los componentes de PLN.
- *Processing resources*: procesadores de PLN.
- *Visual resources*: herramientas gráficas tales como editores o visualizadores de anotaciones.

Las primeras versiones de *GATE* (1997) permitían definir cadenas de procesamiento seleccionando los componentes de PLN sobre un grafo que mostraba las dependencias entre módulos (según las anotaciones que consumían y generaban) de manera similar a algunos de los sistemas de *workflow* presentados en el capítulo 5.

UIMA

A diferencia de *GATE*, *UIMA* utiliza anotaciones fuertemente tipadas (asumen anotaciones totalmente dependientes del dominio). *UIMA* se basa en tener un representación común de las anotaciones (CAS, véase sección 3.4.2) con la que se comunican los diferentes componentes.

UIMA distingue diferentes tipos de componentes:

- *CAS Initializer*: componente para inicializar el CAS.
- *Collection Reader*: componente para leer los documentos de una colección.
- *Analysis Engine*: procesador de PLN.
- *CAS Consumer*: componente para serializar o presentar los resultados.

La arquitectura *UIMA* presentada en el capítulo anterior (sección 5.1) es una de las arquitecturas-sistemas más importantes en PLN, que además ha ido evolucionando desde su inicio. *UIMA Vinci* fue una de la primeras implementaciones que permitían acceder a *UIMA* como un servicio. Los servicios *Vinci* son síncronos, por lo que no permite al cliente hacer otro trabajo durante la transmisión de peticiones.

UIMA Vinci fué reemplazado por *UIMA-AS*, que permite ejecutar procesos *UIMA* de forma asíncrona ofreciendo, así, una mayor escalabilidad de procesos PLN. En *UIMA-AS* se sustituye el llamado *Collection Processing Manager* (CPM), que pasa a estar basado en el estándar JMS. *UIMA-AS* permite encapsular servicios por medio de diferentes motores de análisis, creando así un servicio único y compartido. La tecnología *UIMA-AS* está basada en *Apache ActiveMQ*[9], y es por lo tanto mas flexible que los servicios *UIMA* basados en *Vinci*. *UIMA-AS* crea para cada cliente una nueva cola para las respuestas, y cada petición contiene la dirección única de la cola de respuestas del cliente. El balanceo de carga de los servicios *UIMA-AS* se basa en esta cola compartida de peticiones. Diferentes instancias del servicio *UIMA* requerido pueden ser creadas o destruidas según las necesidades del momento.

Para mejorar la administración de los recursos del *cluster* sobre el que se ejecutan los componentes *UIMA*, se creó más tarde *UIMA DUCC*, un controlador de cluster linux

[9] `http://activemq.apache.org/`

basado en *UIMA-AS* y diseñado para escalar las cadenas de procesos de *UIMA* que requieran un gran *throughput* o computación en tiempo real.

UIMA se ha utilizado en diferentes dominios, como por ejemplo la biología (bionlp-uima[10]) o el procesado de informes médicos (*cTakes, clinical Text Analysis and Knowledge Extraction System*[11]).

En la actualidad han aparecido nuevas arquitecturas como *Argo* (véase sección 5.4) , basadas en *UIMA*, que funcionan según el modelo de *NLP as a service*.

6.1.2 Sistemas de computación distribuida

Esta sección se centra en la escalabilidad de los distintos modelos de ejecución tradicionales (tanto en línea como por lotes o híbridos). Analizaremos los sistemas más utilizados en PLN para conseguir un procesamiento distribuido en *clusters*. Además, describiremos las características específicas utilizadas en la computación basada en *GPU*s.

Sistemas de computación distribuida por lotes

MapReduce es un modelo de programación introducido por *Google* en [DG04] para dar soporte a la computación paralela sobre grandes colecciones de datos. Este modelo de computación paralela consta de dos fases (*map* y *reduce*) cuyos nombres provienen de dos operaciones similares de la programación funcional.

Con la aparición de la computación en la nube, el modelo *MapReduce* se impuso como estándar de facto en la mayoría de ocasiones por la facilidad intrínseca de este paradigma para adaptar algoritmos que no han sido especialmente diseñados para su ejecución en paralelo (a diferencia de otros paradigmas, como *MPI*, que normalmente implican el desarrollo de nuevos algoritmos).

Apache Hadoop[12] es una implementación en código abierto de este paradigma de programación. *Hadoop* fue inicialmente desarrollo por *Yahoo* y actualmente es un proyecto *Apache*.

Se han adaptado diversos algoritmos de aprendizaje usados en PLN a esta arquitectura, permitiendo así su escalabilidad. No obstante, la mayoria de herramientas de PLN no se adaptan correctamente al modelo *MapReduce*, ya que, una vez aprendido el modelo, el etiquetado de documentos sólo utiliza la etapa de *map* (aunque sí se aprovechan la robustez y el balanceo de carga de estos sistemas). Por otro lado, algunas técnicas usadas en PLN se basan en la computación (costosa) de algoritmos iterativos sobre grafos [MR11]. Estos algoritmos basados en grafos tampoco encajan perfectamente en el esquema *MapReduce* ya que requieren diversas iteraciones de tareas *MapReduce* con la consiguiente penalización de coordinación entre nodos y transmitir/guardar los resultados entre iteraciones.

Un ejemplo de integración de diferentes procesadores PLN en una arquitectura *MapReduce* es el proyecto *Behemoth*[13]. *Behemoth* permite ejecutar componentes tanto de *UIMA* como de *GATE* en *Mapreduce*. Sequencialmente, *Behemoth* ejecuta los diferentes componentes de la cadenas de procesamiento lanzando una tarea *MapReduce* para cada componente. Además, el sistema serializa y deserializa el resultado entre cada ejecución de los componentes.

[10] `https://sourceforge.net/projects/bionlp-uima/`
[11] `http://ctakes.apache.org`
[12] `http://hadoop.apache.org/`
[13] `https://github.com/DigitalPebble/behemoth`

Con *Hadoop-YARN*[14] se separa el modelo de computación de la gestión de los recursos permitiendo otros tipos de flujo de ejecución como los que describiremos a continuación.

Sistemas de computación distribuida en *streaming*

Las redes sociales, los dispositivos móviles, *wearables*, los sensores industriales, etc. generan ingentes cantidades de datos de forma constante. A menudo, estos flujos continuos de datos, llamados *streams*, necesitan ser captados, procesados y analizados de forma ininterrumpida, por lo que no se adaptan bien a los sistemas tradicionales de trabajo por lotes. *Apache Storm*[15] es el paradigma de sistema de procesado en línea (tiempo real) de manera distribuida. *Storm* fue desarrollada por *Twitter* para poder definir el flujo de proceso para datos en *streaming*. Así, *Storm* esta diseñado para consumir flujos de datos continuos y procesarlos en un topología arbitrariamente compleja de tareas definidas previamente. Al ser una abstracción sobre el flujo de los procesos, su ejecución puede basarse en diferentes implementaciones de sistemas de colas mencionadas en la introducción de este capítulo.

En general, integrar una herramienta de PLN en un sistema de procesado en línea genérico (p.e. como un procesador, *bolt*, dentro de *Storm*) no es sencillo dadas las restricciones inherentes a la arquitectura, ya que la mayoría de etiquetadores PLN necesitan grandes cantidades de memoria, un tiempo de inicialización alto (carga de modelo) y tienen un *throughput* limitado.

El sistema desarrollado en proyecto *NewsReader* es un ejemplo del uso de *Storm* en arquitecturas PLN (ver sección 5.14). El sistema desarrollado en el proyecto se ejecuta de forma paralela y distribuida, tanto para el procesado por lotes como por *streaming* [Bel+16]. Los componentes PLN se conectan por medio de una topología *Storm* desplegada en diferentes máquinas virtuales en un *cluster*. Además, *NewsReader* desarrolló una serie de *scripts*, llamado *VM from scratch*[16], para crear máquinas virtuales cero con todos componentes necesarios para el procesado PLN distribuido.

Sistemas híbridos de computación distribuida

También han surgido modelos híbridos que intentan combinar lo mejor de las dos aproximaciones, como la llamada arquitectura Lambda que combinan procesado por lotes y el procesamiento en línea. *Apache Flink*[17] es un motor de flujos de datos (*streaming dataflow engine*) cuyo objetivo es facilitar la computación distribuida sobre *streams* de datos, e integra el proceso por lotes como un caso particular de *streams*. Otro ejemplo es *Apache Samza*[18], un entorno de procesado de *streams* construido sobre el sistema de colas *Apache Kafka* para mensajeria y *YARN* para la gestion de recursos.

Otros modelos de computación distribuida

Existen nuevos modelos de computación para poder escalar diferentes tipos de procesos, entre los que cabe destacar *Apache Spark*[19]. El modelo de computación de *Spark* se basa en la descomposición de los procesos en tareas, formando un grafo acíclico dirigido (GAD). Su idea principal es la compartición, de manera trasparente al programador, de la memoria

[14] `http://hortonworks.com/apache/yarn/`
[15] `http://storm.apache.org/`
[16] `https://github.com/ixa-ehu/vmc-from-scratch`
[17] `https://flink.apache.org/`
[18] `http://samza.apache.org/`
[19] `http://spark.apache.org/`

y de los recursos de computación de todos los nodos. El tipo de datos básico de *Spark* es un conjuntos de datos de sólo lectura que se distribuye entre *clusters* de máquinas (los llamado RDDs *Resilient Distributed Dataset*) que han evolucionado hacia formas más fuertemente tipadas como *DataFrames* y *Datasets* (en Spark 2).

Cada tarea de *Spark* crea un GAD de etapas de trabajo para que se ejecuten. Los grafos de tareas creados por *Spark* pueden tener cualquier número de etapas, y es, por lo tanto, un modelo mucho más flexible en comparación con *MapReduce*, que sólo permite dos etapas predefinidas (*map* y *reduce*). A diferencia del esquema *MapReduce*, *Spark* no necesita guardar en disco los resultados obtenidos en las etapas intermedias del grafo, y permite seleccionar las etapas especificas que requieren guardar el resultado. Entre las diferentes etapas, los RDD se pueden almacenan en memoria para que no sea necesario acceder a ellos en disco. Por ejemplo en el caso de algoritmos de aprendizaje automático, donde se itera repetidamente sobre los mismos datos de entrenamiento, el uso de los datos en memoria hace que sea mucho más eficientes.

Las nuevas representaciones tipadas de Spark 2 (*DataSets*) permiten al optimizador *Catalyst* y al motor de ejecución *Tungsten* una mayor optimización de las tareas.

Spark MLlib[20] es un ejemplo de librería de aprendizaje automático sobre *Spark*, e incluye diversos algoritmos utilizados comúnmente en PLN. Por otro lado, existen *wrappers* para integrar herramientas PLN sobre *Spark*: *CoreNLP* [21], *OpenNLP*, etc.

Spark también soporta la computación en línea utilizando el llamado *Spark stremaing*[22] y puede alimentarse de sistemas de colas como *Apache Kafka*, *Flume* o *ZeroMQ*.

De manera similar, pero menos generalizadas, también han surgido nuevos arquitecturas con el objetivo de resolver problemas especificos, como por ejemplo los cómputos sobre grafos. El modelo de computación de *Apache Gigraph*[23] está ligada a la computación de algoritmos sobre grafos y se basa en intercalar la ejecución de diversas iteraciones locales y etapas de sincronización con el resto de nodos (workers).

Últimamente han surgido diferentes propuestas para poder escalar los sistemas de *deep learning*, que típicamente se basan en realizar cálculos vectoriales con matrices de gran tamaño. Estos algoritmos se han desarrollado en paralelo a las mejoras en las tarjetas gráficas (*GPUs*), que ofrecen un rendimiento muy bueno para este tipo de cálculos. Por ejemplo, según la empresa *Nvidia*, una de las desarrolladoras de tarjetas gráficas más importantes del mundo, el modelo *K80* ofrece diez veces más rendimiento que las mejores *CPU*s actuales para aplicaciones científicas.

Sistemas basados en deep learning como *Tensorflow*[24], y su aplicación a PLN *SyntaxNet*, incluyendo analizadores sintácticos como *Parsey McParseface parser*, aumentan su rendimiento notablemente utilizando la potencia de las tarjetas gráficas. Recientemente *CaffeOnSpark*[25], basada en *Spark* y desarollada por *Yahoo labs*, ha empezado a abrir el camino a este tipo de computación distribuida de una manera más eficiente. Casi simultáneamente, *Google* liberó la version distribuida de *tensorflow*[26] basada en grpc y en la reciente conferencia MesosCon'2016, se presentó el soporte de *Mesos* a la gestión de

[20] `http://spark.apache.org/mllib/`
[21] `https://github.com/databricks/spark-corenlp`
[22] `http://spark.apache.org/streaming/`
[23] `http://giraph.apache.org/`
[24] `https://www.tensorflow.org/`
[25] `https://github.com/yahoo/CaffeOnSpark`
[26] `https://www.tensorflow.org/versions/r0.9/how_tos/distributed/index.html`

GPUs en sus *clusters*.

6.2 Distribución de sistemas

El despliegue de una aplicación, tanto en la nube como en un único servidor, implica muchas tareas diferentes que van desde el empaquetado del código o la configuración de máquinas y software, hasta la orquestación de servicios. Hay diversas herramientas para la gestión y configuración para el despliegue de aplicaciones, entre los que cabe destacar *Puppet*[27], *SlatStack*[28], *Chef*[29], y *Ansible*[30]. Estas herramientas de despliegue y configuración, junto con la virtualización de servidores, son los principales componentes que han permitido la computación en la nube.

Describir en detalle estos sistemas está fuera del alcance de este informe. Aún así, diremos, en forma de resumen, que en general la combinación *Puppet+chef* tiene mucho más éxito entre desarrolladores y que, por otro lado, *ansible+saltstack* es muy utilizado en el área de la administración de sistemas. Para el desplegado de software las dos opciones más comunes son *Capistrano*[31] y *Fabric*[32].

Una de las herramientas que propició el desarrollo y uso de técnicas de virtualización de los servidores (necesaria para la coputación en la nube) fue *Vagrant*. *Vagrant* nació con el objetivo de la creación y configuración de entornos de desarrollo virtualizados. Originalmente se desarrolló para el paquete *VirtualBox* y sistemas de configuración tales como *Chef*, *Salt* y *Puppet*.

Actualmente, las principales técnicas utilizadas para la virtualización de servidores son las máquinas virtuales y los contenedores. Las máquinas virtuales son una abstracción del hardware de un ordenador. Permiten emular *CPU*s, dispositivos de almacenamiento, red, etc. *VirtualBOX* y *openVM* son dos de los principales implementaciones usados en la actualidad. Por otro lado, los contenedores representan una virtualización a nivel de sistema operativo (*Operating-system-level virtualization*). Se basa en un método de virtualización donde el núcleo del sistema operativo permite la existencia de múltiples instancias aisladas entre ellas. Estas instancias, a veces llamadas contenedores de software (*software containers*), motores de virtualización (*virtualization engines*) o jaulas (*jails*), se ejecutan desde le punto de vista de los usuarios como si fueran servidores independientes.

Uno de los principales sistemas de contenedores es *Docker*[33], un proyecto de código abierto capaz de automatizar el despliegue de aplicaciones dentro de contenedores de software, proporcionándonos así una capa adicional de abstracción y automatización en el nivel de virtualización de sistema operativo. *Docker* hace uso de las utilidades de aislamiento de recursos del núcleo de linux para permitir que contenedores independientes se ejecuten como instancias únicas, evitando así la elevada sobrecarga en el arranque y mantenimiento de máquinas virtuales. Más adelante se presentan diferentes sistemas de orquestración de containers como Kubernetes o DockerSwarm. Hoy en día existen implementaciones

[27] `https://puppet.com/`
[28] `http://saltstack.com/`
[29] `https://www.chef.io/`
[30] `https://www.ansible.com/`
[31] `http://capistranorb.com/`
[32] `http://www.fabfile.org/`
[33] `http://docker.com`

Docker para muchos procesadores PLN, incluyendo *IXA-pipes*[34], *CoreNLP*[35] o *Freeling*[36].

6.2.1 Computación en la Nube (Cloud Computing)

La necesidad de procesar cantidades cada vez mayores de datos textuales disponibles en las organizaciones públicas y privadas así como también en las redes sociales (Big Data), hace evidente que los ordenadores personales y los *mainframes* equipados con herramientas de PLN estándar no son capaces de hacer frente al procesamiento de texto a gran escala. La opción tradicional comprende comprar y mantener servidores, así como la infraestructura de red en centros de datos propios. A pesar de que las corporaciones tienen los medios y recursos para la adquisición de infraestructuras de hardware para el procesamiento a gran escala y los experimentos con datos masivos, las organizaciones pequeñas y con pocos recursos tales como los laboratorios de universidades y pequeñas compañías tecnológicas están en una posición de desventaja con respecto a los centros privados de investigación, dados los escasos fondos para adquirir hardware para el procesamiento masivo en paralelo. La computación en la nube se ha convertido así en una solución práctica para muchas de estas empresas y organizaciones.

Se han establecido diferentes capas de abstracción/virtualización desde el puro hardware (*IasS*) hasta la aplicación final (*SaaS*). La infraestructura como servicio (*infrastructure as a service, IaaS* o *hardware as a service, HaaS*) es una primera capa de abstracción y es un medio de entregar almacenamiento básico y capacidades de cómputo como servicios estandarizados en la red. Servidores, sistemas de almacenamiento, conexiones, enrutadores, y otros sistemas se concentran (por ejemplo a través de la tecnología de virtualización) para manejar tipos específicos de cargas de trabajo —desde procesamiento en lotes hasta el aumento dinámico de servidores/almacenamiento durante las cargas pico. El ejemplo comercial mejor conocido es *Amazon Web Services*, cuyos servicios EC2 y S3 ofrecen cómputo y servicios de almacenamiento esenciales (respectivamente). Otro ejemplo es *OpenStack* que ofrece diferentes maquinas virtuales (o *Joyent*) con servidores virtualizados para manejar sitios web.

La plataforma como servicio (*platform as a service, PaaS*), es la encapsulación de una abstracción de un ambiente de desarrollo y el empaquetamiento de una serie de módulos o complementos que proporcionan, normalmente, una funcionalidad horizontal (persistencia de datos, autenticación, mensajería, etc.). De esta forma, un arquetipo de plataforma como servicio podría consistir en un entorno conteniendo una pila básica de sistemas, componentes o APIs preconfiguradas y listas para integrarse sobre una tecnología concreta de desarrollo (por ejemplo, un sistema linux, un servidor web, y un ambiente de programación como *Perl* o *Ruby*). Las ofertas de *PaaS* pueden dar servicio a todas las fases del ciclo de desarrollo y pruebas del software, o pueden estar especializadas en cualquier área en particular, tal como la administración del contenido. En este modelo de servicio al usuario se le ofrece la plataforma de desarrollo y las herramientas de programación por lo que puede desarrollar aplicaciones propias y controlar la aplicación, pero no controla la infraestructura.

Ejemplos comerciales son *Google App Engine*, que sirve aplicaciones de la infraestructura *Google*; *Microsoft Azure*, una plataforma en la nube que permite el desarrollo y

[34] `https://github.com/asoroa/ixa-pipes-dockerize`
[35] `https://hub.docker.com/r/motiz88/corenlp/`
[36] `https://github.com/malev/docker-freeling`

ejecución de aplicaciones codificadas en varios lenguajes y tecnologías como *.NET*, *Java* y *PHP*. Servicios *PaaS* como éstos permiten gran flexibilidad, pero puede ser restringida por las capacidades disponibles a través del proveedor.

El Software como un Servicio (*Software as a Service, SaaS*) es un modelo de distribución de software donde el soporte lógico y los datos que maneja se alojan en servidores de una compañía de tecnologías de información y comunicación (TIC), a los que se accede vía Internet desde un cliente. La empresa proveedora TIC se ocupa del servicio de mantenimiento, de la operación diaria y del soporte del software usado por el cliente. La información, el procesamiento, los consumos, y los resultados de la lógica de negocio del software, están hospedados en la compañía de TIC.

El software como servicio se encuentra en la capa más alta y caracteriza una aplicación completa ofrecida como un servicio, por-demanda, vía multitenencia —que significa una sola instancia del software que corre en la infraestructura del proveedor y sirve a múltiples organizaciones de clientes. Las aplicaciones que suministran este modelo de servicio son accesibles a través de un navegador web –o de cualquier aplicación diseñada para tal efecto– y el usuario no tiene control sobre ellas, aunque en algunos casos se le permite realizar algunas configuraciones. Esto le elimina la necesidad al cliente de instalar la aplicación en sus propios computadores, evitando asumir los costos de soporte y el mantenimiento de hardware y software.

SaaS puede describirse como servicios y aplicaciones que están disponibles en línea, donde toda la responsabilidad esta en el proveedor. Básicamente son aplicaciones alojadas remotamente (*hosted application*) que el usuario usa. Este es el caso de *Lucid Chart*[37] o *Gliffy*[38].

Algunos proveedores difuminan la línea entre IaaS y PaaS ofreciendo soluciones híbridas:

- *Amazon Elastic Beanstalk*[39]. *Amazon Elastic Compute Cloud* (*Amazon EC2*) es un servicio web que proporciona capacidad de cómputo con tamaño modificable en la nube. Está diseñado para facilitar a los desarrolladores la computación en la nube escalable basado en web.
- *Azure*[40] es la solución a la computación en la nube de Microsoft. *Azure* contiene una colección de diversos servicios integrados en la nube (análisis, proceso, bases de datos, móviles, redes, almacenamiento y Web).
- *BlueMix*[41] es la propuesta de IBM para dar servicios de computación en la nube (*IaaS, Paas, SaaS*), incluyendo por ejemplo diferentes modelos de contenedores y permitiendo aplicaciones de nube híbrida, así como diferentes servicios desde aplicaciones como *Spark* a servicios de procesamiento de texto (*Alchemy*, o el extractor de relaciones de *Watson*)
- *Rancher*[42] es una plataforma de software en código abierto pensada para alojar contenedores de producción y que soporta tanto *swarm* o *kubernetes* (ver sección 6.3) para la orquestación de servicios.

[37] `http://www.lucidchart.com/`
[38] `http://www.gliffy.com/`
[39] `http://aws.amazon.com/elasticbeanstalk/`
[40] `https://azure.microsoft.com`
[41] `https://console.ng.bluemix.net/`
[42] `http://rancher.com/`

- *OpenShift*[43] es la solución de computación en la nube de plataforma como servicio desarrollada por *Red Hat*. Los desarrolladores pueden usar el sistema de control de versiones *Git* directamente para desplegar sus aplicaciones. *OpenShift* también soporta programas binarios, siempre que se puedan ejecutar en entornos RHEL linux. Esto permite el uso de lenguajes y frameworks arbitrarios. *Origin*, la versión en código abierto de *OpensShift* utiliza *Docker* para la gestión de contenedores y *Kubernetes* para la gestión de grupos de contenedores.

Tipos de Computación en la Nube (Cloud Computing)

La gestión de la computación en la nube es de gran relevancia cuando se trabaja con datos confidenciales. Así junto con estos diferentes niveles de externalización y virtualización/abstracción (en sus diferentes niveles) podemos definir podemos definir tres grandes categorías de funcionamiento de la computación en la nube según donde esté gestionada:

- **Nube pública**: Una nube pública es una nube computacional mantenida y gestionada por terceras personas no vinculadas con la organización. En este tipo de nubes tanto los datos como los procesos de varios clientes se mezclan en los servidores, sistemas de almacenamiento y otras infraestructuras de la nube. Aplicaciones, almacenamiento u otros recursos están disponibles al público a través de el proveedor de servicios, que es propietario de toda la infraestructura en sus centros de datos; el acceso a los servicios sólo se ofrece de manera remota, normalmente a través de internet.
- **Nube privada**: Una nube privada ofrece protección de datos y ediciones a nivel de servicio. Las nubes privadas están en una infraestructura bajo demanda, gestionada para un solo cliente que controla qué aplicaciones debe ejecutarse y dónde. Son propietarios del servidor, red, y disco y pueden decidir qué usuarios están autorizados a utilizar la infraestructura. Se mantiene la privacidad de la información y permite unificar el acceso a las aplicaciones corporativas de sus usuarios.
- **Nube Híbrida**: La nube híbrida combina los modelos anteriores (nube pública y privada). Se es propietario de unas partes (privada) y se comparte otras (publica) aunque de una manera controlada. Las nubes híbridas ofrecen escalado, aprovisionada externamente, a demanda y mantener la seguridad/control sobre de ciertos componentes o datos, pero añaden la complejidad de determinar cómo distribuir las aplicaciones a través de estos entornos diferentes (privado-público).

Actualmente están apareciendo los primeros sistemas que intentan mitigar el vacio actual en nubes hibridas. Por ejemplo CLOUD KOTTA[Bab+16] que mediante OAUT 2[44] y un sistema de roles permite gestionar la privacidad de los datos sobre Amazon EC2. Y que permite encryptar de los datos (server-site) y restringir su acceso mediante una nube provada virtual (Virtual Private Cloud).

6.3 Orquestación de microservicios y componentes

Esta sección presenta las principales herramientas para coordinar y orquestar microservicios, esto es, herramientas para comunicar y hacer extensible los diversos servicios o componentes necesarios entre ellos. *Kubernetes* y *Docker Swarm* son probablemente

[43] `https://www.openshift.com/`
[44] https://oauth.net/2/

dos de las herramientas más utilizadas para desplegar contenedores dentro de un *cluster*. Ambos pueden ser utilizados para administrar un grupo de contenedores y la orquestación de todos los servidores, como si se trataran de un solo contenedor nodo.

6.3.1 Kubernetes

Kubernetes[45] es un sistema de orquestación para contenedores *Docker* en código abierto creado por *Google* en 2015 para la gestión de aplicaciones en contenedores. *Kubernetes* permite programar el despliegue, escalado y la monitorización de contenedores, entre muchas otras más cosas. *Kubernetes* fue diseñado para ser un entorno para la creación de aplicaciones distribuidas de contenedores. El objetivo principal de *Kubernetes* es facilitar la construcción, el funcionamiento y la gestión de sistemas distribuidos. *Kubernetes* de basa en la experiencia de *Google* en organizar y desplegar una aplicación en contenedores de linux (*Google Borg*). *Kubernetes* permite configurar el controlador para mantener activos el mismo número de contenedores y también facilita tanto el escalado vertical "hacia arriba" como "hacia abajo".

6.3.2 Dockers swarm

Docker Swarm[46] es una herramienta nativa para gestionar un *cluster* compuesto de contenedores *Dockers*. *Swarm* es una iniciativa de *Docker* para lograr que un conjunto de máquinas se comuniquen con el exterior por medio de una sola API. Con *Docker Swarm* tenemos un nodo master, *Swarm Master*, que será el responsable del cluster; también requiere de otros nodos que debe ser accesible por el nodo maestro que contendrán cada uno de ellos uno o más contenedores.

6.3.3 Mesos

Apache Mesos[47] es una capa de abstracción que representa un centro de datos, similar a Omega de Google[48] (conocida anteriormente como *Google Borg* y que fue la base de *Kubernetes*). *Mesos* nació en la Universidad de Berkeley, y es ahora un proyecto *Apache* de código abierto utilizado por *Twitter* para administrar sus centros de datos.

El objetivo de *Mesos* es facilitar la administración de todos los recursos (*CPU*, memoria, almacenamiento) de las máquinas (tanto físicas como virtuales) que están en un centro de datos, para así habilitar sistemas distribuidos elásticos y tolerantes a fallos que sean fáciles de construir y se ejecuten eficientemente. En *Mesos* todos los procesos están dentro de un contenedor y el sistema decide a qué recursos asignar a los procesos, dependiendo de la necesidad de esa tarea (memoria, *CPU*, etc.). Con esto, toda la carga es distribuida y se evita la sobrecarga de servidores por algún pico en algún servicio en particular.

Mesos permite coordinar tanto trabajos en *Hadoop* como en *MPI*. Además, soporta *Docker* en forma nativa. *Mesos* permite programar aplicaciones para el *datacenter* sobre estas capas de abstracción, de manera que los desarrolladores no se preocupen tanto por la parte de infraestructura. En la parte más baja está la capa física, es decir todos los servidores (con su sistemas operativos correspondientes) y después está el sistema

[45] `http://kubernetes.io`
[46] `https://docs.docker.com/swarm/`
[47] `http://mesos.apache.org/`
[48] ver `http://www.wired.com/2013/03/google-borg-twitter-mesos/`

de ficheros distribuido, donde se instala *Mesos*. Sobre esta capa se deben instalar los *frameworks* de sistemas distribuido (*Hadoop*, *Spark*, etc.), que son como un puente entre nuestra aplicación y *Mesos*.

Dentro del un *cluster* de *Mesos* tendremos los nodos maestros (coordinados a través de *Zookeeper*, para mantener siempre un único nodo maestro), los nodos esclavos (*slaves*) y los *frameworks*. Los nodos esclavos son los encargados de ejecutar las tareas de los *frameworks*; éstos, a su vez, reportan su estado directamente al nodo maestro activo, que se identifica a través de *Zookeeper*. Los *frameworks* son instalados en los nodos maestros, y a través de ellos se usan los recursos de los esclavos. Hay diversos *frameworks* predefinidos y se pueden desarrollar *frameworks* propios tanto en *Java*, *Python* y C++[49].

La gestión del uso de GPUs en un *cluster* todavia es un tema por resolver (especialmente si tenemos un cluster donde no todos los nodos tiene GPU o con diferentes usuarios/tipos aplicaciones GPU-no GPU). La ejecucion de los procesos en containers (donde se podria limitar el uso de la GPU asignado a cada proceso) podria ser la solucíon ideal. Recientemente *Mesos* ha incorporado la gestión de *GPUs* lo que permite un mayor control de la ejecución distribuida de trabajos que se basen en GPU (como *Tensoflow*). En YARN actualmente solor se podrian usar etiquetas (para distinguir los nodos con GPUs).

[49] `http://mesos.apache.org/documentation/latest/app-framework-development-guide/`

7. Plataformas PLN

Como se ha visto en la sección 6.2.1, hoy en día existen soluciones de computación en la nube para la ejecución de trabajos desde cualquier lugar y en cualquier momento. Muchas de estas soluciones están especialmente diseñadas para el problema de procesamiento de grandes cantidades de datos y la democratización del acceso a los recursos informáticos para la experimentación. Algunas plataformas ofrecen servicios con costo (*pay per use*) que pueden ser más ventajosa que la adquisición, instalación y mantenimiento de infraestructuras de hardware complicadas y costosas.

En este capítulo veremos las principales plataformas PLN que se adhieren al paradigma de la computación en la nube y ofrecen a usuarios y pequeñas y medianas empresas una solución para el tratamiento PLN de grandes cantidades de texto.

7.1 GATECLoud

GATECould.net [Tab+12] es una plataforma desarrollada en Universidad de Sheffield en el Reino Unido para el procesamiento de texto a gran escala en la nube. Se basa en el denominado paradigma "plataforma como servicio" (*PaaS*, ver sección 6.2.1) que libera a los desarrolladores de PLN de las cuestiones relacionadas con los detalles de implementación de la plataforma. *GATECould* se basa en la infraestructura *GATE* que se describe en la sección 4.3. Ofrece una serie de componentes de PLN para crear cadenas de procesamiento adaptadas (Reconocedores de entidades nombradas, clasificadores de opinión, analizadores sintácticos, etc.). También libera al desarrollador de cuestiones relacionadas con el formato de texto / documento a tratar en la plataforma a través de la utilización de formatos estándar como JSON o XML. *GATECloud* es diferente de las plataformas de procesamiento de texto que siguen el paradigma "software como servicio" (SaaS, ver sección 6.2.1), pues en *GATECloud* el usuario puede llevar sus propias aplicaciones personalizadas a la nube. Estas pueden ser creadas y probadas en un ordenador personal usando *GATE* antes de subirlas a la plataforma. También permite el procesamiento

de texto masivo en el que todo un conjunto de datos se pueden cargar en la nube para el procesamiento (SaaS requeriría el envío de textos a través de la comunicación HTTP estándar sobre la red).

GATECloud aprovecha la potencia de cálculo que ofrece la infraestructura de la nube de Amazon. Como resultado, los servicios que se ofrecen son de pago por uso. Aquí comentaremos brevemente algunos de los aspectos de la implementación de la infraestructura.

- Se utiliza la infraestructura *GATE* de PLN: una biblioteca extensible y un entorno de desarrollo que ofrece recursos para la gestión de datos (el documento *GATE*, anotaciones, ontologías, etc.), léxicos y gramáticas para el reconocimiento de entidades nombradas, programas de aprendizaje automático, etc. (ver sección 4.3). También utiliza *GATE Mimir* [Tab+15], una herramienta para indexar y buscar más anotaciones lingüísticas, y *GATE Teamware* [Bon+13] (ver sección 7.5), una infraestructura para la anotación colaborativa de documentos. Debido a su complejidad, estos dos componentes son ofrecidos de forma predeterminada en la infraestructura nube.
- La arquitectura *GATECould* comprende: (i) una aplicación Web que se ocupa de los usuarios, ofreciendo seguridad sobre el conjunto de datos que se sube a la nube. También proporciona una interfaz para la gestión de servidores y puestos de trabajo, (ii) los Servicios Web de Amazon: servidores virtuales, y (iii) un servidor Apache especialmente configurado para proporcionar persistencia a los servidores del usuario.
- Además de *Mimir* y *GATE Teamware* que están preinstalados, el usuario puede reservar un servidor para iniciar el procesamiento de sus trabajos. Cuando ya no se necesita el servidor, el usuario puede destruirlo. Sólo las horas utilizadas son facturadas al usuario.
- El procesamiento de una tarea en la infraestructura requiere subir una colección de documentos a la nube, subir una aplicación *GATE*, y ejecutarla. El usuario es informado sobre el estado del trabajo y cuando este finaliza. El trabajo a ejecutar puede ser cualquier aplicación *GATE* que se haya creado con la plataforma *GATE* y posteriormente exportado a la nube. Los datos producidos por el sistema se mantiene en el espacio de usuario en la nube.
- La infraestructura ofrece seguridad a través de canal cifrado con SSL.

7.2 TextFlows

TextFlows [Per+16] es una plataforma de código abierto para la minería de texto basado en Web que soporta el diseño y ejecución de *workflows* definidos por el usuario. Se basa en una arquitectura cliente-servidor y se distribuye de forma gratuita bajo una licencia de software MIT.

El servidor de la arquitectura (implementado en el código abierto basado en *Django*) comprende: (i) una base de datos relacional (*PostgreSQL*) para almacenar las especificaciones de los *workflows* (gestionado por una librería de software llamada *PySimpleSoap*), (ii) trabajadores a los que se les puede asignar tareas para su ejecución, y (iii) un agente a cargo de la asignación de tareas a los trabajadores y del control de su ejecución (la distribución de tareas se lleva a cabo gracias a un sistema de gestión de colas llamado *Celery*).

El lado del cliente de TextFlows comprende una interfaz gráfica de usuario implemen-

tada para navegadores Web. Tiene las funcionalidades de una interfaz gráfica de usuario moderna, tales como seleccionar, "arrastrar y soltar" para facilitar la definición de flujos.

En TextFlows, los flujos de trabajo son procedimientos gráficos ejecutables. Pueden ser diseñados mediante la composición de *widgets* con representaciones gráficas (los parámetros para los *widgets* se pueden introducir también mediante su GUI). Los flujos de trabajo se almacenan en el servidor y sólo pueden ser accedidos por sus creadores (a menos que estos los hagan públicos). Todos los *widgets* disponibles para componer los flujos de trabajo se organizan en clases jerárquicas (tokenizadores, etiquetadores, etc.). Existen *widgets* que proporcionan funcionalidades de minería de textos, manipulación de datos, visualización y descripción de servicios web. En lo que concierne la adquisición de datos, varios componentes están disponibles para cargar documentos desde carpetas locales o a través de servicios web o bases de datos, también de procesos de *crawling* de dominios Web específicos, o *scraping* de datos de resultados de búsquedas web.

7.3 OpeNER

En el marco del proyecto europeo *OpeNER* del séptimo programa marco (FP7) de la Unión Europea se ha desarrollado una *suite* de herramientas abiertas para realizar varias tareas de PLN en seis lenguas: inglés, español, italiano, holandés, alemán, y francés. Las herramientas se centran en el área de reconocimiento y clasificación de entidades, NED, y análisis de sentimientos.

Las herramientas de *OpeNER* [Age+13; GCR13] producen una salida en formato XML siguiendo el formato *KAF* (versión previa a *NAF*, ver sección 3.5.1). Para facilitar su uso directo por los usuarios finales, una serie de servicios Web permiten encadenar y escalar la ejecución de los módulos *OpeNER* en la nube de *Amazon*.

7.4 PANACEA

PANACEA [Poc+12] es una plataforma de procesamiento de lenguaje natural basada en servicios Web desarrollada en el proyecto europeo del mismo nombre. El objetivo de esta plataforma es proveer software distribuido para facilitar la creación de recursos lingüísticos. *PANACEA* se basa en el concepto de flujos de trabajo, secuencias de invocaciones a procesadores implementados como servicios Web.

PANACEA está diseñada como una plataforma de servicios web que permite al usuario encadenar diferentes servicios de una forma amigable para crear cadenas de procesamiento complejas. Los servicios Web disponibles en la plataforma provienen de diversas organizaciones, tanto compañías como universidades, que los mantienen. La plataforma está implementada con herramientas del proyecto *myGrid*. Los servicios Web se despliegan usando el software *Soaplab* que está bien adaptado para manejar servicios que requieren mucho tiempo de ejecución.

PANACEA contiene un registro de servicios Web disponibles implementado usando una adaptación de *Biocatalogue*, un registro que se encarga de la indexación de descriptores de servicios y permite búsquedas y etiquetado.

Los *workflows* en *PANACEA* están implementados a través de Taverna (ver sección 5.6) [Hul+06; Oin+07]. *PANACEA* también ofrece una red social en la que los usuarios pueden compartir sus *workflows* con el resto de usuarios. Aquí se pueden anotar los flujos

de trabajo con informaciones de tipo diverso (descripción, licencia, etc.) que facilita su posterior re-utilización. Esta funcionalidad ha sido implementada con una adaptación de la herramienta *myExperiment*.

7.5 GATE Teamware

GATE Teamware [Bon+13] es una plataforma Web abierta para la anotación colaborativa de documentos. Está diseñada para trabajar en tareas de anotación de corpus de una complejidad media/alta. Si bien la plataforma está basada en el sistema *GATE* (ver sección 4.3) que tiene una interfaz para la anotación de documentos destinada a usuarios especialistas, *GATE Teamware* oculta las complejidades del sistema a través de una interfaz simple que está destinada a usuarios no especialistas. La plataforma tiene tres tipos de usuarios: anotadores que son los encargados de ejecutar tareas especificas de anotación siguiendo guías pre-establecidas, *managers* encargados de definir las tareas de anotación y *workflows*, y administradores encargados de cuentas de usuarios, privilegios, etc.

La plataforma ofrece servicios para procesar documentos en varios formatos y exportarlos en XML *stand-off* (ver sección 3.2). Los servicios de anotación automática permiten la ejecución de cadenas de procesamiento basadas en algoritmos proporcionados por *GATE*. El servicio de autentificación verifica los privilegios de los usuarios mientras que el manejador de *workflows* se encarga de la creación, eliminación, ejecución y monitorización de los flujos de trabajo. Existen tres tipos principales de interfaces: la del anotador, la de los adjudicadores (que concilian las anotaciones), y la de los administradores; cada una con funcionalidades específicas.

7.6 TextServer

TextServer [PT15] es una plataforma para el PLN en la nube de la Universitat Politècnica de Catalunya que destaca por las siguientes características:

- Ofrece soluciones escalables y procesamiento eficiente gracias al uso de una plataforma de hardware (*SunGrid Cluster* de más de 1000 procesadores) que permite el procesamiento masivo en paralelo.
- Soporta procesamiento multilingüe gracias a la plataforma de PLN *Freeling* [PS12] (vease sección 4.5).
- Permite un acceso simple a los procesadores via una interfaz Web.
- Hace posible la reproducibilidad de experimentos .

Para satisfacer las necesidades de diferentes usuarios, *TextServer* ofrece dos interfaces para el procesamiento de documentos. Una interfaz Web permite enviar varios documentos para ser procesados. Por otro lado varios programas cliente están disponibles que hacen posible la conexión al cliente y el envío de documentos. Los procesos pueden ser ejecutados en lotes o en modo interactivo. Los procesos se ejecutan en la plataforma de computación masiva de acuerdo con la planificación realizada por un planificador de tareas. La plataforma siempre tiene procesadores libres de manera a poder atender nuevos pedidos de procesamiento. Nuevas instancias de procesadores son creadas o destruidas según la carga de trabajo en un momento determinado. *TextServer* ofrece varios servicios, entre los cuales se encuentran:

- Identificador del idioma del documento.
- Tokenización y segmentación de oraciones.

- Transcripción fonética.
- Análisis morfológico.
- Etiquetado léxico.
- Desambiguación del sentido de las palabras.
- Reconocimiento de entidades nombradas.
- Chunking.
- Parsing de dependencias y etiquetado de roles semánticos.
- Resolución de la correferencia.
- Generación de grafo semántico.

Para utilizar los servicios de *TextServer* se debe suscribir a la plataforma como usuario. Dependiendo del caso los servicios pueden ser libres o de pago. La disponibilidad de servicios multilingües varia según de que procesador se trate. Por ejemplo, el identificador de correferencias solo está disponible en español e inglés, mientras que la tokenización y el etiquetado léxico están disponibles para varios idiomas.

7.7 Google Cloud Natural Language API

Google Cloud Natural Language API[1] es la propuesta de *Google* para dar servicios de PLN nube dentro de uns entorno completo de computación en la nube[2]. Actualmente está en fase beta pero ya incluye procesadores para inglés, castellano y japonés que permiten realizar el enlace de entidades a *Wikipedia*, analizar la polaridad de los textos o realizar el análisis sintáctico. Desde hace ya tiempo, *Google* ofrece uno de las mejores servicios en la nube de traducción automática[3]

7.8 Servicio en la nube (BlueMix)

BlueMix[4] es la plataforma abierta de IBM para crear aplicaciones empresariales personalizadas. *BluexMix* ofrece como servicios en la nube diversos componentes de PLN que forman parte de *Watson* (extracción de relaciones, conversión de texto a voz y de voz a texto, systemas de dialogo, etc.). *BlueMix* también permite usar *OpenWhisk*, una tecnología de código abierto desarrolla por IBM enfocada a la programación orientada a eventos en la nube.

7.9 Apache Stanbol

Apache Stanbol[5] es un sistema de código abierto para desarrollar aplicaciones en la nube por medio de componentes reusables. En *Stanbol* la ejecución el flujo de trabajo se define de forma declarativa por medio de un grafo dirigido que conecta diferentes componentes. *Stanbol* incluye algunos componentes PLN[6] (pos, NERC, chunking) para el procesado básico de textos.

[1] `https://cloud.google.com/natural-language/`
[2] `https://cloud.google.com/products/`
[3] `https://cloud.google.com/translate/`
[4] `http://www.ibm.com/cloud-computing/bluemix/es-es/`
[5] `https://stanbol.apache.org/`
[6] `https://stanbol.apache.org/docs/trunk/components/enhancer/nlp/`

7.10 Meaning Cloud (Sngular Meaning)

Meaning Cloud[7] (anteriormente *Daedalus*) es una empresa que ofrece servicios en la nube para el procesado PLN. Contiene diferentes servicios PLN (análisis del sentimiento, morfológico o sintáctico) para desarrollar aplicaciones de minería de texto complejas.

7.11 AlchemyAPI

AlchemyAPI[8] es una empresa que ofrece servicios PLN, generalmente basados en herramientas de aprendizaje automático, que desarrollan hasta 12 tareas PLN diferentes (extracción de términos, reconocimiento de entidades, análisis de sentimiento, etc.). *AlchemyAPI* es una librería PLN que dispone de una API para los lenguajes de programación más importantes (*Python*, *PHP*, etc.). Al ejecutar la aplicación del usuario, las llamadas a la librería se ejecutan de forma remota y distribuida en la nube. El servicio *Alchemy* es de pago y usualmente el usuario debe abonar una cantidad fija de dinero por cada llamada a la API.

[7] `http://www.meaningcloud.com`
[8] `http://www.alchemyapi.com/`

Bibliografía

[BKL09] Steven Bird, Ewan Klein y Edward Loper. *Natural language processing with Python*. O'Reilly Media, 2009 (véase página 49).

[Cun+11] Hamish Cunningham y col. *Text Processing with GATE (Version 6)*. 2011. ISBN: 978-0956599315. URL: `http://tinyurl.com/gatebook` (véase página 46).

[Fel98] Christiane Fellbaum. *WordNet*. Wiley Online Library, 1998 (véase página 14).

[MR11] Rada Mihalcea y Dragomir Radev. *Graph-based natural language processing and information retrieval*. Cambridge; New York: Cambridge University Press, 2011. ISBN: 9780521896139 (véase página 74).

[Age+13] Rodrigo Agerri y col. "OpeNER: Open Polarity Enhanced Named Entity Recognition." En: *Procesamiento del Lenguaje Natural* 51 (2013), páginas 215-218 (véase página 85).

[Bab+16] Yadu N. Babuji y col. "A Secure Data Enclave and Analytics Platform for Social Scientists". En: *CoRR* abs/1610.03105 (2016). URL: `http://arxiv.org/abs/1610.03105` (véase página 80).

[Bon+13] Kalina Bontcheva y col. "GATE Teamware: a web-based, collaborative text annotation framework." En: *Language Resources and Evaluation* 47.4 (2013), páginas 1007-1029 (véase páginas 84, 86).

[Bri+12] Doug Briesch y col. "Training and Evaluating a Statistical Part of Speech Tagger for Natural Language Applications using Kepler Workflows". En: *Procedia Computer Science* 9 (2012), páginas 1588-1594 (véase página 68).

[Car+03] J. Carletta y col. "The NITE XML Toolkit: flexible annotation for multi-modal language data". En: *Special issue on Measuring Behavior. Behavior Research Methods, Instruments, and Computers* 35.3 (jun. de 2003) (véase página 28).

[DZ11] Mita K Dalal y Mukesh A Zaveri. "Automatic text classification: a technical review". En: *International Journal of Computer Applications* 28.2 (2011), páginas 37-40 (véase página 16).

[Eps+12] Edward A Epstein y col. "Making watson fast". En: *IBM Journal of Research and Development* 56.3.4 (2012), páginas 15-1 (véase páginas 27, 28).

[FL03] Scott Farrar y D Terence Langendoen. "A linguistic ontology for the semantic web". En: *Glot International* 7.3 (2003), páginas 97-100 (véase página 25).

[GR13] Maarten van Gompel y Martin Reynaert. "FoLiA: A practical XML format for linguistic annotation - a descriptive and comparative study". En: *Computational Linguistics in the Netherlands Journal* 3 (dic. de 2013), páginas 63-81. ISSN: 2211-4009 (véase página 30).

[Goy+16] Ankit Goyal y col. "Natural Language Processing Using Kepler Workflow System: First Steps". En: *Procedia Computer Science* 80 (2016), páginas 712-721 (véase página 68).

[Hul+06] D Hull y col. "Taverna: a tool for building and running workflows of services". En: *Nucleic Acids Res* 34 (jul. de 2006), páginas 729-732 (véase páginas 67, 85).

[IR04] N. Ide y L. Romary. "International standard for a linguistic annotation framework". En: *Natural Language Engineering* 10.3-4 (2004), páginas 211-225. ISSN: 1351-3249 (véase página 17).

[Kan+11] Y. Kano y col. "U-Compare: a modular NLP workflow construction and evaluation system". En: *IBM Journal of Research and Development* 3 (2011) (véase página 63).

[Liu12] Bing Liu. "Sentiment analysis and opinion mining". En: *Synthesis lectures on human language technologies* 5.1 (2012), páginas 1-167 (véase página 15).

[Mil+91] G. A. Miller y col. "Five Papers on WordNet". En: *Special Issue of the International Journal of Lexicography* 3.4 (1991), páginas 235-312 (véase página 23).

[NLS11] Heike Neuroth, Felix Lohmeier y Kathleen Marie Smith. "TextGrid - Virtual Research Environment for the Humanities". En: *International Journal of Digital Curation* 2 (2011), páginas 222-231 (véase página 69).

[PGK05] Martha Palmer, Daniel Gildea y Paul Kingsbury. "The Proposition Bank: An Annotated Corpus of Semantic Roles". En: *Computational Linguistics* 31.1 (mar. de 2005), páginas 71-106. ISSN: 0891-2017. DOI: 10.1162/0891201053630264. URL: http://dx.doi.org/10.1162/0891201053630264 (véase página 15).

[Per+16] Matic Perovek y col. "TextFlows: A visual programming platform for text mining and natural language processing". En: *Science of Computer Programming* 121 (2016), páginas 128-152. ISSN: 0167-6423 (véase página 84).

[Tab+12] Valentin Tablan y col. "GATECloud.net: a platform for large-scale, open-source text processing on the cloud". En: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 371.1983 (2012). ISSN: 1364-503X (véase página 83).

[Tab+15] Valentin Tablan y col. “MíMir”. En: *Web Semant.* 30.C (ene. de 2015), páginas 52-68. ISSN: 1570-8268 (véase página 84).

[Vos+16] Piek Vossen y col. “NewsReader: using knowledge resources in a cross-lingual reading machine to generate more knowledge from massive streams of news”. En: *Knowledge-Based Systems* (2016) (véase página 12).

Índice alfabético

A

B

C

D

E

F

G

H

I

J

K

L

M

V

W

Y