UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

SERVICIO WEB DE IDENTIFICACIÓN Y CLASIFICACIÓN DE ENTIDADES NOMBRADAS

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

AUTOR: Ana Belén Megías Sánchez.

TUTOR: Paloma Martínez Fernández.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

INDICE DE CUADROS

AGRADECIMIENTOS

En primer lugar me gustaría agradecer a Paloma, mi tutora, todo el apoyo que me ha dado y su paciencia con mis idas y venida.

A mis amigos, Natalia y Javi, que tantas horas de biblioteca hemos compartido.

A mis compañeros de trabajo, Edu, Marta, Pio y Cristina, por esas 'ayuditas' claves en momentos de desesperación absoluta y todo el ánimo que me han dado.

A mi familia política, Angelines, Ino y Maria, que tan bien me cuidan.

Sobre todo a mi padre, que con tanto esfuerzo ha conseguido que llegue a donde he llegado y a mi madre porque sus 'tupper' me han hecho creer que estábamos más cerca cuando nos separaban 400km, porque es mi madre y se merece todo.

Y por supuesto a David, mi medio yo. Por su 'venga Anita, que no te queda nada' en mis momentos de bajón, por estar siempre a mi lado, por su atención… simplemente por estar ahí. Gracias Amore!

¡Mil gracias de corazón!

CAPÍTULO 1. INTRODUCCIÓN

1.1. MOTIVACIÓN.

El conocimiento es uno de los recursos más preciados para el ser humano y existe en el mundo en volúmenes inmensos de información en forma de lenguaje natural: libros, periódicos, internet etc. La gran cantidad de información disponible dificulta su manejo y organización. Y, aunque suene paradójico, encontrar información específica y útil a veces se convierte en una tediosa tarea.

Es por esto que las computadoras juegan un importante papel en el procesamiento de la información. En la década de los cuarenta empezaron a surgir los primeros brotes sobre PLN derivado de la necesidad de procesar todo el volumen de información disponible. Uno de los problemas a tratar era poder distinguir y extraer elementos de información relevante relacionada con nombres de personas, lugares u organizaciones de un documento. Los sistemas de extracción de entidades (EEN) y más concretamente subáreas como como NER (Named Entity Recognition) y NEC (Named Entity Clasification), realizan la tarea de buscar información muy concreta en colecciones de documentos, detectar información relevante, extraerla y etiquetarla en un formato adecuado para su procesamiento automático. A esa información con creta se le conoce como Entidades Nombradas (NE).

Una Entidad Nombrada es una palabra o conjunto de palabras que contienen un nombre de persona, localización, organización, fechas, cantidades, etc.

En la actualidad existen gran variedad de herramientas comerciales y entornos de desarrollo especializados en la extracción de entidades y muchas de ellas ofrecen muy buenos resultado para el etiquetado en castellano.

También hemos de tener en cuenta que, en la era digital en la que vivimos, contar con herramientas de identificación y clasificación en entidades a través de Internet se ha convertido en casi una necesidad. En los últimos años la gran mayoría de las aplicaciones se ofertan en forma de Servicios Web, lo que también facilita la integración en numerosas aplicaciones y sistemas.

Es así como nace este trabajo; con el objetivo de acoplar tres herramientas de PLN para procesar textos en castellano, obteniendo las entidades de cada una de ellas por separado, marcar todas ellas en el texto según un mapping definido y ofrecerlo en forma de Servicio Web.

1.2. OBJETIVOS.

La finalidad de este proyecto es construir un Servicio Web donde tres recursos analicen un texto y se obtenga como salida el mismo texto anotado según tipos (personas, localizaciones u organizaciones). Para llevar a cabo este trabajo se marcaron ciertos objetivos necesarios.

A nivel de conocimientos, estudiar y analizar diferentes herramientas para la identificación y clasificación de EN, así como los técnicas y sistemas de extracción. Conocer también cómo funcionan estos sistemas por dentro, de qué manera identifica y clasifican las entidades y los etiquetados que manejan para analizar sus salidas con el fin de diseñar un mapping común.

A nivel técnico, los objetivos marcados estaban en seleccionar un lenguaje de programación estructurado que facilitara la tarea de acoplar los tres recursos y un entorno de desarrollo de aplicaciones que soportara este lenguaje. Finalmente Java fue el lenguaje seleccionado y Eclipse el entorno de desarrollo. Eclipse facilita la tarea a la hora comunicarse con otros servicios y sistemas como, Tomcat y Axis,

seleccionados como servidor y motor de Servicios Web y así poder ofrecer un sistema de EEN en forma de servicio.

1.3. ESTRUCTURA DE LA MEMORIA.

El documento se estructura en siete apartados principales. Cada uno de ellos aborda un tema asociado al desarrollo del proyecto.

En primer lugar, en la introducción se da una breve explicación sobre el proyecto en general, siguiendo con el planteamiento del problema que explica por qué la necesidad de hacer este proyecto y el objetivo, es decir, que queremos.

En segundo lugar se introduce el origen de PLN y las EN para después entrar más en detalle en las diferentes técnicas y sistemas de identificación y extracción de EN.

Continúa en tercer lugar analizando herramientas comerciales y entornos de desarrollo de PLN desde varios puntos de vista.

En capítulo 4 se entra en detalle de los requisitos del sistema así como la arquitectura software seleccionada para el desarrollo del Servicio Web. También se detalla el cómo se ha diseñado e implementado el sistema y que funcionalidades presenta.

La quinta parte corresponde al plan de pruebas que se han llevado a cabo para probar el correcto funcionamiento del servicio web y comprobar la cobertura del etiquetado. Tras esta sección se expone las fases del proyecto, así como la consecución de los objetivos marcados.

Por último se exponen las conclusiones y posibles trabajos futuros.

Finalmente aparecen unos anexos en los que se explican todos los pasos para instalar el entorno de trabajo incluyendo la instalación de Eclipse, Apache Tomcat y Axis2 y las tecnologías y lenguajes utilizados en el desarrollo del Servicio Web.

También forman parte de este documento la bibliografía consultada, una lista de acrónimos y definición de términos usados a lo largo del documento y un índice de figuras.

CAPÍTULO 2.
ESTADO DE LA CUESTIÓN

En este capítulo se hará mención y analizarán, de manera resumida, las principales técnicas y herramientas para la extracción de entidades así como entornos de desarrollo de aplicaciones PLN.

2.1. INTRODUCCIÓN

La **Inteligencia Artificial** (IA) surge a partir del influyente trabajo en 1.950 de Alan Turing con su artículo Computing Machinery and Intelligence, en el que propuso una prueba concreta para determinar si una máquina era inteligente o no, su famosa *Prueba de Turing*.

Y de esta área de la computación es donde nace el procesamiento del lenguaje natural (PLN). Esta subárea de la Inteligencia Artificial tiene como objetivo el procesamiento automático del lenguaje. En este sentido, es una disciplina aplicada a diversidad de tareas, como extracción de información, recuperación de información, resumen automático de documentos, segmentación, clasificación de documentos, indexación de documentos, creación de plantillas, manipulación de vocabularios controlados o diccionarios, etc.

Entre las taras de PLN encontramos la Extracción de Información (EI). Es habitualmente definida como el proceso de estructurar y combinar selectivamente los datos que están explícita o implícitamente en uno o más documentos en lenguaje natural" [3]. Aunque también podemos encontrarla definida como el proceso de derivar datos no ambiguos, a partir de textos en lenguaje natural, en servicio de algún tipo de necesidad de información precisa previamente especificada [46]. En cualquier caso, se trata de un proceso que implica la clasificación semántica de unidades de información.

Esas unidades de información son "**Entidad Nombrada**" (Named Entity-NE). El término fue acuñado en la sexta conferencia MUC [1][40] (Message Understanding Conference). Estas conferencias fueron iniciadas en 1987 bajo el patrocinio de la agencia norteamericana de defensa DARPA, con el objetivo de fomentar los métodos de Extracción de Información (EI) a partir de textos no estructurados. El objetivo era clasificar semánticamente la información de interés, que sobre todo en las primeras conferencias, tenía tintes militares. Ya en 1995, con la 6ª conferencia MUC, se ve la importancia de la identificación de nombres de personas, organismos y localizaciones, y de expresiones numéricas como el tiempo y las cantidades. Esta tarea es denominada Reconocimiento de Entidades Nombradas (en adelante Reconocimiento de Entidades o NER), constituyéndose como un importante campo dentro de la IE.

En investigaciones previas ya se habían tratado problemas similares. Éste es el caso del trabajo presentado en 1991 por Lisa F. Rau et al. [2], donde se describe un sistema para extraer y reconocer nombres de compañías mediante el uso de heurísticas y reglas elaboradas ad hoc a partir de observaciones. Sin embargo, es desde finales de los 90 cuando se acelera el ritmo de las publicaciones en este ámbito y comienzan a surgir además foros de evaluación relacionados con esta y otras tareas de extracción de información: HUB [41], IREX: Information Retrieval and Extraction Exercise [42], CoNLL: Conference on Natural Language [43], ACE: Automatic Content Extraction [44], etc.

2.2. TÉCNICAS DE EXTRACCIÓN DE ENTIDADES [3].

Las técnicas aplicadas en el reconocimiento de entidades han evolucionado desde la elaboración de patrones manuales, obtenidos a través de recursos etiquetados a mano, hasta reglas obtenidas automáticamente mediante técnicas de aprendizaje.

En la figura 1 se muestra un esquema de las técnicas de EI.

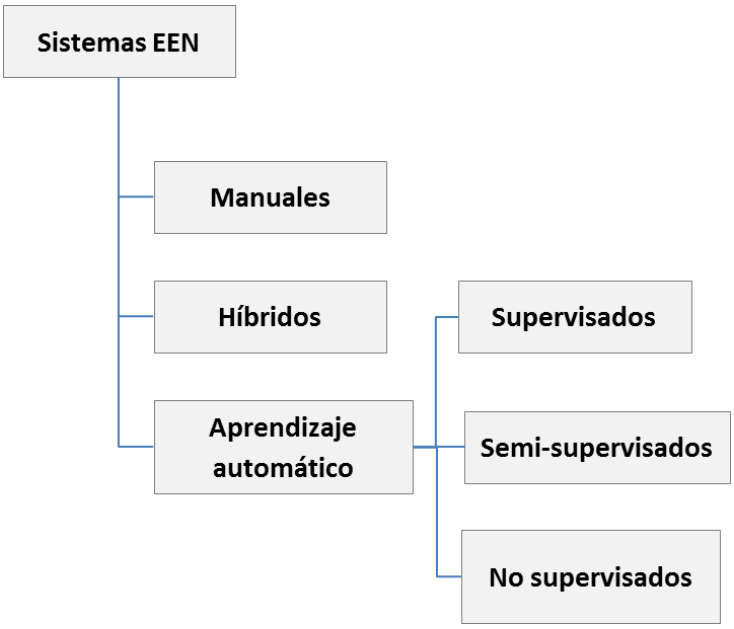


FIGURA 1: clasificación sistemas EEN.

A continuación se hace una breve incursión sobre las diferentes técnicas de extracción de información y sus características más relevantes.

2.2.1. SISTEMAS BASADOS EN REGLAS (HANDCRAFTED).

Estos sistemas son construidos a mano. Ayudados por un conjunto de reglas y heurísticas que guían el proceso de etiquetado, están basados en el conocimiento de los especialistas que los diseñan.

Los recursos más utilizados para estos sistemas son las expresiones regulares, conjuntos de reglas y Gazetteers (listas de palabras). Generalmente están compuestos por conjuntos de patrones con características gramaticales, sintácticas y ortográficas en combinación con diccionarios.

Los sistemas manuales tienen la ventaja de poder obtener muy buenos resultados. Una gran mayoría de ellos alcanza una precisión del 90%, aunque para mayor rendimiento sea necesario definir manualmente un conjunto de reglas por cada par lenguaje/dominio. Tanto es así que, estos tipos de modelos tienen mejores resultados para dominios restringidos que los modelos basados en aprendizaje automático. Son capaces de detectar entidades complejas cosa que presenta un problema para los sistemas automáticos.

Sin embargo, este tipo de enfoques carece de la habilidad de hacer frente a los problemas de robustez y portabilidad, tan necesarios a día de hoy. Cada texto o fuente nueva requiere una revisión de las reglas para mantener un rendimiento óptimo. Lo que quiere decir que cada vez que haya un nuevo dominio y/o lenguaje, habría que reescribir el sistema. Además hay que añadir que el coste de mantenimiento podría ser muy elevado.

2.2.2. SISTEMAS BASADOS EN APRENDIZAJE AUTOMÁTICO.

Tiene como objetivo desarrollar técnicas que sean capaces de generalizar comportamientos a partir de una información estructurada que es suministrada en forma de ejemplos.

Las investigaciones sobre algoritmos de aprendizaje automáticos fueron motivadas a partir de la posibilidad de no necesitar gente especializada en el campo para examinar los datos y encontrar patrones o reglas asociados a éstos. De hecho, actualmente, la tendencia de la EEN es utilizar métodos de aprendizaje automático, puesto que se adaptan con mayor facilidad a distintos dominios.

No obstante, hay factores que pueden afectar a los resultados obtenidos por estos sistemas como pueden, ser un mal etiquetado de los corpus o las limitaciones propias del método de aprendizaje utilizado. De ahí que los sistemas basados en aprendizaje supervisado hayan obtenido mejores resultados que aquellos basados en aprendizaje no supervisado.

- **Métodos de aprendizaje supervisados**

Para utilizar algoritmos de aprendizaje supervisado es necesario contar con un corpus donde las entidades estén anotadas para que sirvan como ejemplo en el entrenamiento. La tarea de etiquetar el corpus requiere un esfuerzo considerable, no siendo necesario un grado de especialización alto. Sin embargo, para el personal que define los conjuntos de las reglas en los sistemas manuales ha de ser considerable.

Dentro de los sistemas que utilizan aprendizaje automático, encontramos los basados en reglas y los basados en métodos estadísticos.

Los **métodos basados en reglas** son de fácil comprensión, desarrollo y expansión, además de rápidos y útiles para tareas controladas, como la extracción de números de teléfono, códigos postales de correos electrónicos, etc. Están formados por dos partes, una colección de reglas y las políticas necesarias para controlar el disparo de esas reglas.

Las reglas pueden ser aprendidas mediante técnicas de aprendizaje automático, a partir de ejemplos etiquetados en textos no estructurados o bien con métodos **Handcrafted.** El objetivo de este método es conseguir un conjunto de reglas tales que posean una buena precisión y cobertura en nuevos documentos. Existen dos modos:

- **Bottom-Up**: como el algoritmo (LP)2, parten de una regla muy específica con el 100% de precisión y cobertura mínima consiguiendo generalizarse a través del aprendizaje.

- **Top-Down**. Basado en algoritmos como FOIL y WHISK parten de una regla muy genérica y se especializan con el aprendizaje.

Los **métodos basados en estadísticas** [5] convierten la tarea de extracción en un problema de descomposición del texto no estructurado etiquetando las partes de la descomposición bien de forma conjunta o independiente. Según este tipo de descomposición se puede distinguir tres tipos de métodos: **Token-level**, **Segment-level** y **métodos basados en gramáticas**. Todos ellos reciben como entrada parte de un texto no estructurado compuesto por Tokens y de un conjunto de tipos de entidades que se quieren extraer de él.

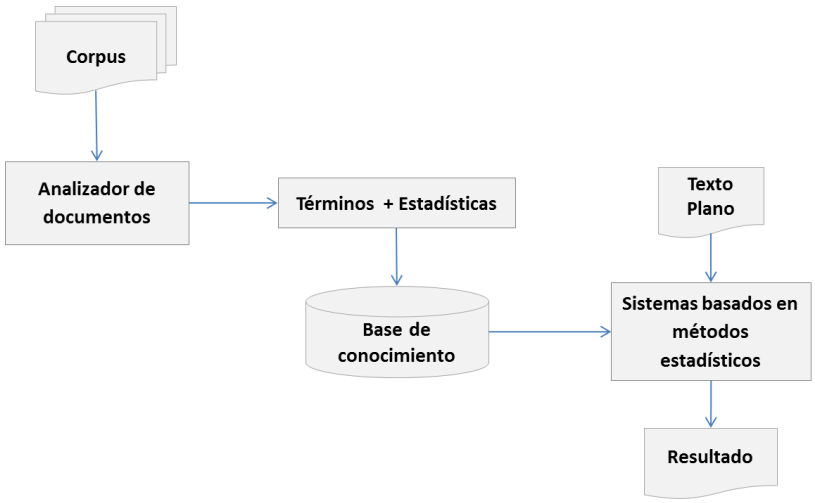


FIGURA 2: Métodos basados en estadísticas.

En los modelos Token-level el texto no estructurado se trata como una secuencia de tokens, reduciendo la extracción a asignar a cada token una etiqueta de entidad. Cuando todos los tokens están etiquetados, las entidades se marcan como tokens consecutivos de la misma etiqueta entidad. Las propiedades que se utilizan en estos modelos suelen ser propiedades de palabra, ortográficas y de aparición en diccionarios.

Para la clasificación la pareja token-entidad se suelen utilizar los modelos ocultos de Harkov (HMM-Hiden Markov Model-primer sistema NER estadístico de alto rendimiento), etiquetadores de entropía máxima (MEMM-Maximum Entropy Markov), modelos condicionales de Markov (CMM-Conditional Markov Models) y los CRF (Conditional random fields) que presentan mejores resultados que los métodos lineales (como SVM-Suppor Vector Machine) tanto teórica como empíricamente.

En los **modelos segment-level**, la salida es una secuencia de segmentos, chunks donde cada uno define una entidad. Las características se definen sobre

segmentos compuestos por múltiples Tokens que forman una cadena completa de entidad. Esto le permite utilizar atributos más potentes que los token-level. Así, además de utilizar los token-level, se pueden usar parámetros con similitud a una entidad existente en una base de datos y la longitud del segmento de la entidad.

Estos modelos son efectivos para fases bien formadas (fbf) o estructuras del lenguaje natural. Si tratan de explotar la estructura global del texto fuente fallan, al no seguir el formato de sentencia bien formada. Esto se puede remediar con el uso de **modelos basados en gramáticas libres** dado que son más efectivos para estructuras fuera que se encuentren al margen de fbf. Ocurre porque usan un conjunto de reglas de producción para expresar la estructura global de la entidad dando como salida un árbol de parseado o análisis.

A pesar de que estos modelos basados en segmentos y en gramáticas proporcionan toda la flexibilidad de los CRF's, no son muy populares debido al aumento de coste en la inferencia y entrenamiento. Tampoco los métodos puramente automáticos basados en datos para la inducción de reglas lo son debido a la limitada existencia de datos etiquetados. Es por ello que los sistemas basados en reglas más exitosos hacen uso de métodos híbridos, donde se extraen reglas semilla de los datos etiquetados para después modificar o afinar esas reglas.

- **Métodos de aprendizaje semi-supervisados**

Cómo ya se ha mencionado anteriormente, el etiquetado manual de un corpus es tarea costosa, por lo que no siempre se puede contar con la disponibilidad de suficientes muestras de entrenamiento para los clasificadores. Sin embargo, existen cantidad de textos sin etiquetar a los que se pueden acceder de forma fácil y económica. El objetivo del aprendizaje semi-automático es combinar muestras etiquetadas y no etiquetadas para mejorar los clasificadores. La técnica más popular es el "bootstrapping" [6], el cual requiere un conjunto de semillas que usa al principio para buscar oraciones o instancias que coincidan con estas semillas.

Algunos sistemas de este tipo exploran redundancias entre las características internas y contextuales [7][8] de las entidades. Un ejemplo, una pista interna de palabra es el hecho de que una palabra capitalizada, comenzando con 'Sr.',es probable

que sea un tributo de la clase PERSON. Por otro lado, una pista contextual es que una palabra seguida por 'S.A.' sea probablemente del tipo ORGANIIZATION.

Otro método semi- automático es el llamado **Aprendizaje Activo** (Active Learning -AL), donde es el propio sistema el que proporciona al usuario los candidatos para que los corrija o etiquete, de modo que puedan ser utilizados como nuevas semillas para reentrenar el modelo. Existen diversas técnicas para la selección de los ejemplos a etiquetar [45]:

- **Uncertainty sampling:** se seleccionan las instancias con menos certeza de que sean válidas.
- **Query by Committee:** cuando el aprendizaje se realiza con varios algoritmos a la vez, se seleccionan las instancias donde más desacuerdo exista entre ellos.
- **Density-Weighted Methods:** se seleccionan las instancias no sólo con menos certeza de validez, sino que además son representativas de la distribución de entrada. De este modo se trata de evitar la selección de outliners a la que son propensos los métodos anteriores.
- **Expected Model Change:** se seleccionan aquellas instancias que más puedan influir en el modelo.
- **Variance Reduction and Fisher Information Ratio:** se eligen las instancias bajo el criterio general de minimizar la varianza.
- **Estimated Error Reduction:** se seleccionan las instancias que minimizan el error esperado.

- **Métodos de aprendizaje no supervisados**

Aunque este tipo de métodos no necesitan ejemplos de entrenamiento, no son muy populares para la extracción de entidades debido a que los sistemas que lo usan no han alcanzado el nivel de desempeño de los supervisados. Por ello, los sistemas que aplican este enfoque usualmente no son completamente no-supervisados, sino

que tienden a ser híbridos. Generan un conjunto de reglas a partir de la combinación de módulos de aprendizaje supervisado y no supervisados con el objetivo es construir representaciones de los datos.

Lo métodos no supervisados pueden ser fácilmente portados a diferentes dominios y lenguajes. El enfoque típico para este tipo de aprendizaje es el clustering, dado que trata de extraer ENs de clusters basados en la similitud del contexto. Básicamente las técnicas recaen en recursos y patrones léxicos y en estadísticas calculadas en corpus grandes no etiquetados [9].

Nadeau et al, usa una estrategia no supervisada usando Gazetteers generados automáticamente y desambiguando después para reconocer tres tipos de entidades: ORG, LOC, PER. No llega a ser competitivo con sistemas supervisados puesto que alcanza un factor de rendimiento de un 70%.

Cucchiarelli y Velardi usaban el resultado de una técnica basada en corpus de aprendizaje para clasificar nombres propios no reconocidos. Este sistema es complementario a los métodos actuales de reconocimientos de EN. El objetivo es mejorar, sin esfuerzos manuales adicionales, la robustez de cualquier sistema mediante el uso de conocimiento contextual más refinado y mejor explotado en una etapa relativamente tardía del análisis. El método es particularmente útil cuando el sistema EN debe ser rápidamente adaptado a otro lenguaje o dominio.

2.2.3. SISTEMAS HÍBRIDOS.

La estrategia detrás de los sistemas híbridos consiste en combinar las fortalezas de los sistemas manuales con aquellos basados en aprendizaje automático para reducir sus debilidades. La mayoría de los sistemas analizados son, hasta cierto punto, híbridos pues aunque se basen en modelos de aprendizaje automático, hacen uso de recursos lingüísticos como Gazetteers o listas de palabras. A continuación se describen algunos ejemplos de sistemas híbridos:

- **LTG [10],** hace uso de evidencia interna y externa basándose en la idea de que ciertas cadenas tienen una estructura que sugiere que son ENs pero no de qué tipo. La idea principal es retrasar la clasificación final de un NE hasta que esa pieza de información contextual sea encontrada y las ENs sean desambiguadas.

- **MENE [11],** es otro sistema con buen rendimiento. Utiliza un clasificador de entropía máxima para combinar salidas de varios sistemas hechos a mano y obtiene resultados superiores a los obtenidos por cada uno de dichos sistemas independientes. El etiquetado de ENs se aborda como un etiquetado de secuencias donde varias características (internas, externas, locales, globales) son desarrolladas y combinadas

- **ME & HMM [12],** presentado por Srihari et al., combinan ME (entropía máxima), HMM (Hiden Markov Model) y reglas gramaticales hechas a mano. Aunque cada método tiene sus debilidades, la combinación de ellos dio como resultado un etiquetador de alta precisión, además incluyen Gazetteers internos**.**

- **Cruz [13],** presentó un enfoque híbrido para español. Este sistema combina clasificadores secuenciales markovianos con un conjunto de heurísticas en forma de expresiones regulares**.**

En fin, la tarea de la EEN es abordada por muchos enfoques, tanto manuales, automáticos o híbridos. Según Feldmal, se puede ver el problema de EEN como un problema de clasificación. Un aspecto muy importante de esto es que hay 2 tipos de evidencia complementarios, la interna que se toma dentro de un EN y la externa, la cual provee el contexto en el que aparece ese nombre.

A pesar de que la tendencia a usa formalismos de aprendizaje automático crece constantemente es conveniente la integración de conocimientos lingüísticos a estos debido a la simplicidad y generalidad que puede tener el mismo. En este aspecto el uso de Gazetteers parece inevitable, y varios experimentos realizados evidencian que el uso de estos recursos es positivo.

CAPÍTULO 3.
TRABAJOS RELACIONADOS

3.1. HERRAMIENTAS COMERCIALES

Cada vez existen más aplicaciones comerciales enfocadas al análisis automático de textos con el objetivo de explotar todas las posibilidades de extraer el contenido existente tanto en documentos escritos como en la web.

En la era digital que estamos viviendo, cada vez es más habitual contar con un software como servicio de manera que se puedan integrar fácilmente en diversos sistemas o aplicaciones. En la tabla 1 se muestra un resumen de las principales herramientas comerciales ofrecidas en modo de servicio.

	Términos de uso del API	Idiomas	Numero de Tipo de entidades
OpenCalais	50.000 pet. / día. Mostrara Logo. 100K	Inglés, francés, español.	39
Zemanta	1.000 pet. /día. Mostrara Logo. 8K	Inglés.	6
AlchemyAPI	30.000 pet. /día. Uso comercial definido	97 idiomas.	26
MeaningCloud	40.000 pet/día gratuito.	Español, Ingles, Francés, Italiano, Portugués, Catalán	30 aprox.
Yahoo [18]	5.000 pet. /día. Uso comercial	inglés	20 aprox. (no documentado).
OpenAmplify [19]	5.000 pet. /día. Uso comercial	inglés	Más de 22

TABLA 1: características de principales aplicaciones de extracción de entidades.

Aunque existen numerosos foros donde se evalúan las funcionalidades de NER ofrecidas por diferentes herramientas, ninguna de ellas es de manera formal. No se siguen métodos rigurosos que ofrezcan datos fiables sobre las evaluaciones lo que produce que existen discrepancias entre las distintas evaluaciones.

La evaluación llevada a cabo por ViewChange.org en Entity extraction & Content API Evaluation [14] concluye que Zemanta [15] es la mejor herramienta para la extracción de entidades seguida de OpenCalais [16] y AlchemyAPI [17]. La preferencia de Zemanta sobre OpenCalais se centra en el hecho de que en su evaluación encontraron que Zemanta ofrece mejores resultados en términos de calidad y cantidad de entidades nombradas desambiguadas. En ese mismo blog, hay comentarios de mucho usuarios que citan a OpenCalais como mejor herramienta en cuanto a extracción de entidades, ya que consideran que Zemanta no es bueno detectando muchas entidades que si encuentra OpenCalais (no está adaptada para otros idiomas, es capaz de identificar un número muy reducido de tipos de entidades, etc).

3.1.1. ZEMANTA.

Actualmente la aplicación es un plugin que hay que instalar en un navegador Web y funciona para gran variedad de plataformas de blogs existentes.

Zemanta permite enriquecer las entradas que escribimos en un blog: el servicio analiza lo que escribimos y nos ofrece cuatro tipos de recomendaciones: imágenes, artículos relacionados, enlaces internos y etiquetas (palabras clave). Se puede conseguir asociarlos a nuestra entrada de blog con solo pulsar sobre ellos, consiguiendo añadir más información a la publicación sin necesidad de ir a buscarla.

Ofrece el uso de su servicio Web a través de su API para abrir la posibilidad de utilizar sus capacidades para otras aplicaciones. Este API consiste en un mensaje REST vía HTTP PORT.

Las peticiones al servicio web son sencillas, como entrada de la solicitud se ha introducir un texto libre HTM o TXT (no mayor de 8 Kb) y como salida ofrece un archivo en XML, JSON, RDF/XML entre otros. El servicio más completo de esta herramienta retorna el texto de entrada con links, imágenes, palabras clave, etc., e indica que grado de confianza tiene cada uno. Las entidades retornadas son desambiguadas y proporcionan varios enlaces a bases de datos abiertas tales como FreeBase o DBpedia.

Para alcanzar esta funcionalidad, Zemanta analiza el texto libre usando algoritmos de PLN semánticos que comparan de forma estadística los resultados con su base de datos pre-indexada. Utilizan una combinación de técnicas de aprendizaje automático y el feedback de los usuarios bloggers para el entrenamiento constante del motor y así poder mejorar las recomendaciones.

3.1.2. ALCHEMYAPI.

Es una aplicación web que proporciona a los propietarios de contenido y a los desarrolladores web una amplia gama de herramientas de anotación de meta-datos y

análisis de contenidos. De este modo, puede ser utilizada para enriquecer websites, blogs, sistemas de gestión de contenidos o aplicaciones webs semánticas.

Utiliza técnicas estadísticas de NLP y algoritmos de aprendizaje automático. Para exponer la información semántica del contenido, AlchemyAPI realiza extracción de entidades nombradas, categorización de documentos, detección de idioma, extracción de datos estructurados, etc., para lo cual utiliza técnicas estadísticas de NLP y algoritmos de aprendizaje automático.

Según su web, la funcionalidad de identificación de idioma de AlchemyAPI es a día de hoy la más robusta en el mundo de la industria, pues soporta 97 idiomas diferentes aunque presenta mejores resultados de etiquetado para inglés. Además, a partir de una web en HTML, textos o imágenes escaneadas es capaz de identificas más de 25 tipos diferentes de entidades (personas, empresas, ciudades, organizaciones, continentes, etc.).

El API tiene tres variantes: Web API, Text API y ATML API y han de utilizarse según se quiera procesar el contenido de una URL pública, de un fichero de texto o un documento HTML. En la solicitud ha de indicarse qué tipo de formato de salida se desea (XML, JSON, RDF o micro-formatos), si se quiere desambiguar, co-referencias, etc.

3.1.3. OPENCALAIS.

Es uno servicios Web de Thomson Routers que permite la extracción de entidades, eventos de textos libres en inglés, francés y español aunque para inglés ofrece más funcionalidades que para el resto de idiomas.

Entre las funcionalidades que ofrece para español solamente implementa la extracción de entidades y la puntuación (Score) de la relevancia de entidades; el resto como desambiguación, etiquetado social (simula el etiquetado que realiza una persona) o categorización de documentos solamente están implementadas para inglés.

Ofrece un API sencillo y gratuito para que los usuarios puedan hacer uso de las funcionalidades de OpenCalais. Este API puede ser usado como mensaje SOAP, REST

vía HTTP POST, lo cual es muy útil para la integración de dicho módulo con tecnologías WEB. Como entrada puede recibir distintos formatos, TXT/HTLM, TEXT/HTMLRAW, TEXT/XML, TEXT/RAW y para procesarlo ejecuta una limpieza que elimina todas aquellas partes del documento que no sean necesarias. A continuación identifica el idioma y analiza el texto para lo que utiliza un sofisticado sistema de reglas.

Para la creación de las reglas se utilizan elementos basados en diferentes niveles PNL, desde tokenización del texto, análisis morfológico y etiquetado POS hasta parseado poco profundo e identificación de frases nominales y verbales. Así mismo también hace uso de diccionarios propios que combinados con un enfoque de descubrimiento de nuevas entidades, permite que OpenCalais sea capaz de identificar nuevas entidades y desambiguarlas dependiendo del contexto en el que aparezca.

3.1.4. MEANINGCLOUD

El API *Topics Extraction* [20] es uno de los componentes que ofrece MeaningCloud para trabajar con entidades. Al API te permite ajustar el las preferencias de anotación para obtener la información más ajustada a las necesidades de cada petición.

El API permite como entrada un TXT (plano, HTML, XML), un URL (HTTP o FTP) o DOC (RTF, PDF, herramienta de Microsoft Office, OpenDocument, etc) . Como salida se obtiene XML,JSON, HTML. Además el tipo de entidades, los diccionarios que se desean utilizar, nivel de desambiguación entre otras funcionalidades son totalmente configurables.

Cuando se envía la petición POST realiza la segmentación del texto en unidades (Tokenizador). Se marcan como entidades candidatas aquellas unidades que aparezcan en alguno de los diccionarios de entidades del sistema, bien tal cual o como una variante. Y finalmente, si para una entidad se tiene más de una candidata, se realiza una desambiguación basada en heurísticos, como la frecuencia de aparición en el texto de la entidad, la presencia de marcadores discursivos (*a+LOCATION* y *artículo+ORGANIZATION*, *aMadrid* se desambigua como la ciudad y *el Madrid*

como equipo de fútbol) o desambiguación geográfica por contexto (según las referencias geográficas que aparezcan).

De forma esquemática, se puede concretar que las entradas léxicas pueden acompañarse de los siguientes rasgos semánticos:

<Tipo de entidad><temática><remisión><Info geográfica><relación>

Gracias a que ofrecen un SDK (Software Delepment Kit), se puede integrada fácilmente en cualquier aplicación que se necesite.

3.1.5. JRC-NAMES.

JRC-Names[21] está compuesto por varias listas de palabras llamadas Trigger Words, esto es, listas de patrones de palabras, sentencias o títulos que normalmente acompañan a entidades candidatas. Estas listas se obtienen a partir de aprendizaje automático, bootstrapping o manualmente desde fuentes de internet. Las palabras son agregadas ordenadamente a las listas con la finalidad de elaborar un repositorio que contenga en cada línea, la palabra y la frecuencia de aparición con una determinada entidad.

Este método es relativamente simple y puede parecer costoso comparado con los métodos de aprendizaje automático basado en la copian patrones de textos. Pero cuenta con numerosas ventajas; no necesita herramientas lingüísticas como analizadores morfológicos, es modular y puede adaptarse fácilmente a otros idiomas proporcionando las trigger words en el idioma seleccionado y las reglas para él.

Para el tipo de entidad ORGANIZATION, esta débilmente desarrollado. Estas entidades son reconocidas si alguna de las palabras que acompañan a la entidad candidata es una organización de las que aparece en la lista (organización, club, banco, etc.). Adicionalmente, un clasificador Bayesian es entrenado para decidir que entidades de personas y organizaciones conocidas han de añadirse a las listas.

Identifica alrededor de 1000 nombres nuevos al día lo que significa quer JRC-Names cuenta con más listas de más de 1.15 millones de entidades y 200.00 variantes.

3.1.6. TEXTRAZOR.

TextRazor [22] ofrece a sus usuarios un API desarrollada bajo cliente Phyton, lo que le proporciona sencillez en el procesamiento de textos, rapidez y potencia. Precisamente gracias a su sencillez puede ser integrada en cualquier lenguaje de programación.

Actualmente cuenta con en API gratuita siempre que los textos a procesar no sean mayores de 100KB. Esta API procesa de manera recurrente dos textos en unos pocos milisegundos y hasta 500 en un día e identifica automáticamente el idioma de su contenido. Soporta francés, alemán, italiano, español, portugués y ruso aunque solo para Inglés tiene el conjunto completo de funciones disponibles. Para necesidades más amplias en su página se pueden contratar los planes de los que disponen.

Al API se le envía una petición HTTPS y el servicio marca las entidades y proporciona sinónimos y URL's de webs dónde se puede encontrar más información sobre la entidad en cuestión. También permite añadir manualmente nombres de personas, compañías, localizaciones o incluso reglas personalizadas de clasificación.

EL alto rendimiento de esta herramienta tiene como base varios algoritmos de aprendizaje automático entrenados con un corpus propio el cual contiene una mezcla de estilos de texto a partir de noticias y medios de comunicación social. A esto añadimos una gran base de información creada a partir de la extracción de entidades de millones de webs, creando así un diccionario con miles de combinaciones para cada entidad. Entre sus entrañas también hay un etiquetador estadístico que identifica identidades nunca antes mencionadas y es capaz de extraer el contexto del texto completo trás haber sido procesado.

3.2. ENTORNOS PARA DESARROLLO DE APLICACIONES DE PLN.

Por lo general, las aplicaciones de PLN se implementan como un conjunto de tareas pequeñas organizadas en un flujo: la salida de una tarea es la entrada de la

siguiente. Muchas de esas tareas se repiten en diferentes aplicaciones: separadores de Tokens y oraciones, analizadores léxico-morfológicos, analizadores sintácticos, etc. Gran parte del esfuerzo de construir una aplicaciones de PLN no radica en la implementación de los módulos requeridos –la mayoría pueden encontrarse como software de código abierto-, sino en lograr que estos diferentes módulos se comuniquen e intercambien datos.

Teniendo en cuenta este problema, la comunidad de PLN ha estado trabajando en enfoques para aumentar la interoperabilidad de las soluciones, a modo de facilitar su integración en nuevas aplicaciones. Esto incluye la definición de estándares de representación de objetos (textos, transcripciones manuscritas, voz grabada), la configuración de las diferentes tareas, construcción del flujo de ejecución, etc.

La mayoría de las arquitecturas de PLN actuales, como GATE o UIMA, no solo utilizan estas definiciones para la representación de sus sujetos de análisis, sino que además definen mecanismos para la integración de módulos desarrollados en diferentes lenguajes y su ejecución como un flujo, accediendo eventualmente a recursos externos (diccionarios, archivos de n-gramas, etc.). En el proceso de desarrollo de una nueva aplicación sobre estas arquitecturas, el investigador debe instalarlas y configurar la plataforma seleccionada, junto con las diferentes herramientas que se requieren para cada una de las tareas.

3.2.1. UIMA.

La arquitectura UIMA [23] (Unstructed Information Architecture), una plataforma libre para la creación, integración y desarrollo de aplicaciones a gran escala que maneja información no estructurada.

Fue originalmente desarrollada por IBM, y es ahora un proyecto de la fundación Apache. Su objetivo principal es proveer un entorno común en donde los desarrolladores pueden colaborar en la creación e intercambio de soluciones PLN, aprendizaje automático y recuperación de información.

A pesar de que UIMA trabaja con distintos tipos de fuentes, aquí nos centramos en el procesamiento de textos para extracción de entidades nombradas. Cada componente de UIMA procesa la información en su entrada, generando como salida

información estructurada. La plataforma permite combinar distintos motores de análisis para crear aplicaciones complejas de procesamiento de lenguaje natural, aislando los algoritmos de PLN de la tarea propia, tales como la comunicación de componentes, intercambio de información, manejo de recursos, etc.

Siguiendo los pasos de TIPSTER, UIMA utiliza un enfoque referencial, guardando anotaciones las cuales refieren a todo o parte del texto original. Estas anotaciones asocian etiquetas a porciones de texto, identificando las posiciones de inicio y fin. Toda la información que los componentes UIMA comparten están en un formato llamado (Common Analysis System), y la arquitectura provee un contenedor de objetos que proporciona una API para su manipulación y modificación. Las etiquetas están definidas sobre un sistema de tipos dado por el usurario, y a su vez, los tipos tienen atributos que cada componente publica a los demás, agregándolos a un índice.

3.2.2. BALIE/YOONAME.

Es un software de libre distribución desarrollado por la Universidad de Ottawa (Canadá) [24]. Está diseñado en Java y soporta inglés, francés y español, alemán y rumano. Entre sus usos destacan la tokenización de textos, el reconocimiento de EN o la identificación de idioma a partir de un texto. Para llevar a cabo esta última tarea descrita utiliza dos técnicas:

- **Análisis de Frecuencia de Palabras de Parada** (Stop Words Frequency Analysis). Consiste en, dado un texto, contar el número de palabras del tipo (the, a,of, etc.), y las asocia al lenguaje que más se aproxime.

- **Análisis de Frecuencia de N-gramas**. Un n-grama es una subsecuencia de n elementos de una secuencia dada. Un n-grama de tamaño 2 se denomina \bigrama"o \digrama"; de tamaño 3, \trigrama"; de tamaño 4 _o más se denomina \n-grama" o \modelo de Markov de orden (n - 1)". Es un buen indicador del idioma al que nos enfrentamos. Por ejemplo, un texto con una alta frecuencia de bigramas \wh" es más probable que se trate de un texto en inglés que en francés. Una ventaja de la utilización de éstas técnicas es que no

hace falta ningún diccionario, pero puede resultar un problema si queremos analizar textos cortos, ya que en muchos casos puede resultar ambiguo.

YooName [25] es la evolución de Balie, por lo que también está implementado en Java, pero no es de licencia libre. Es un software basado en el aprendizaje semisupervisado, lo que consiste en una serie de técnicas de aprendizaje automático que utiliza datos de entrenamientos, tanto etiquetados como no etiquetados. Es capaz de identificar nueve categorías distintas de EN, entre las que se encuentran:

- **Persona**: celebridades, personajes, vocaciones, nombre, apellido, etc.
- **Organización**: compañía, militar, gobierno, equipo, etc.
- **Localización**: Ciudad, estado, país, formación montañosa, ríos, etc.
- **Instalaciones:** rascacielos, escuelas, aeropuertos, carreteras, calles, etc.
- **Productos**: vehículos arte, medios de comunicación,, aras, comida, etc.
- **Eventos**: juegos, guerras, huracanes, crímenes, conferencias, etc.
- **Naturaleza:** insectos, animales, vegetales, minales, etc.
- **Unidad**: moneda, mes, día de la semana, medida, etc.
- **Miscelánea**: Enfermedad, religión, lengua, premio, etc.

Como característica destacable dado su aprendizaje, la aplicación es extensible, por lo que puede reconocer nuevas entidades si se implementan. Además el mantenimiento es automático, realiza periódicamente conexiones a internet para mantener el software actualizado.

3.2.3. GATE.

GATE [26] (General Architecture for Text Engineering) es una infraestructura open-source basada en Java la cual está diseñada para desarrollar y reutilizar componentes de software en la tarea de resolver problemas en el dominio del Procesamiento de Lenguaje Natural. También permite construir, anotar o etiquetar corpus y evaluar las aplicaciones generadas conectándose a una base de datos de textos.

GATE es una arquitectura, un framework y un ambiente de desarrollo. El framework GATE está compuesto de una biblioteca central y un conjunto de módulos reusables (como el tokenizador, el delimitador de oraciones, y el analizador morfosintáctico o Part-Of-Speech Tagger).

Es importante aclarar, sin embargo, que algunos componentes de GATE son dependientes del idioma inglés, en particular los módulos de Extracción de Información.

Los módulos de GATE se clasifican en Recursos de Lenguaje, Recursos de Procesamiento y Recursos Visuales. Los Recursos de Lenguaje son entidades tales como lexicones y corpus. Los módulos de procesamiento son primordialmente algorítmicos, como parsers y tokenizadores. Los Recursos Visuales son los componentes usados en la interfaz gráfica. De esta forma GATE separa los datos, los algoritmos y las formas de visualizarlos para facilitar la reutilización de módulos.

Cuando se ejecutan recursos de procesamiento (como tokenizadores, parseadores, etc.) que operan sobre textos, producen información acerca de éstos. Por ejemplo, un tokenizador, a cada palabra le asigna un tipo (token-type): word, number, punctuation, etc; un etiquetador morfosintáctico, le asigna una categoría gramatical (proper noun, verb, etc). Esta información que se produce a partir del texto se representa dentro de GATE como un conjunto de anotaciones que consiste en:

- Un **ID**, es decir, una identificación única en el documento al que la anotación se refiere
- Un **type**, el cual denota el tipo de anotación. Los diferentes recursos de procesamiento usualmente generan anotaciones de distintos tipos.
- **StartNode y EndNode**. Estos nodos denotan la posición inicial y la posición final que la palabra que se está anotando ocupa en el texto original.
- un conjunto de **features** en la forma de pares atributo/valor que proporcionan información adicional acerca de la anotación.

Dentro de la interfaz gráfica de GATE, las anotaciones de un texto procesado se pueden ver haciendo doble click en el nombre del texto. Además se puede ver el texto anotado en formato XML.

3.2.4 BRAT.

BRAT [27] Es una herramienta desarrollada por la universidad de Manchester y está especialmente diseñada para anotar textos estructurados, donde las entidades o conjunto de estas siguen una estructura concreta, es decir, siguen unos patrones de manera que los textos pueden ser procesados automáticamente (con sistemas no supervisados) mediante reglas para posteriormente ser interpretados por una máquina. Aun así, implementa diferentes características para anotaciones manuales.

Como características especiales, cabe mencionar que BRAT además de anotar entidades aporta las relaciones n-arias (Figura 3) que pudiese haber entre ellas y nos permite añadir o ampliar comentarios sobre entidades.

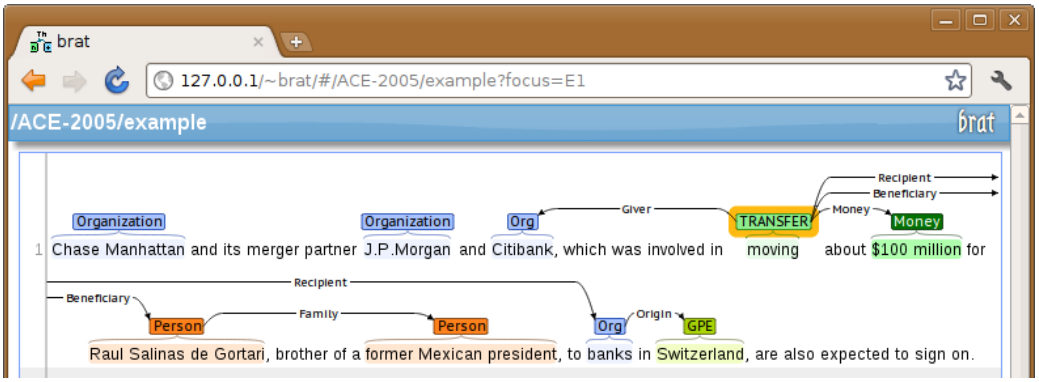


FIGURA 3: etiquetado BRAT.

A la hora de procesar un documento, BRAT no anota sobre el texto, sino que crea un nuevo archivo con extensión .ann en el cual guarda las posiciones de cada entidad junto con el tipo; por lo tanto habrá un archivo.txt que contendrá el texto y otro archivo.ann donde aparecerán en cada línea un ID que corresponderá a la entidad, a que tipo pertenece, la posición inicial de la entidad y la final, las relaciones que pudiesen haber entre ellas y la entidad en cuestión.

Detecta tipos de persona, organización, localización y miscelánea y algunos subtipos (continentes, países,...). Está disponible para 13 idioma aunque solamente 4 son totalmente gratuitos (danés, portugués, sueco y holandés).

CAPÍTULO 4.
DESCRIPCIÓN DEL MODELO

En este capítulo se abordará la descripción informática de la propuesta, así como su diseño e implementación. Para ello se tratará en detalle las partes que componen la aplicación así como la metodología seleccionada para la implementación.

El objetivo de este trabajo es la construcción de un servicio web compuesto por tres herramientas de identificación y clasificación de entidades nombradas (MeaningCloud, JRC-Names, Balie) al que se le proporcionará un texto y debe producir el mismo texto anotado por una, dos o las tres herramientas según los tipo: PERSON, ORGANIZATION y LOCATION. Puesto que se el sistema se implementado como un servicio web, la salida podrá ser reutilizada por otros procesos o aplicaciones.

Además se ha diseñado una aplicación web, donde traduce el texto anotado a HTML para que la interfaz gráfica pueda interpretarlo y muestre las entidades de cada tipo resaltadas en distintos colores.

4.1. REQUISITOS DEL SISTEMA

El Servicio Web debe proporcionar la funcionalidad propia de una herramienta de etiquetado de texto al cual se le ha añadido una funcionalidad extra diseñando una interfaz web para facilitar al usuario la usabilidad.

Diferenciamos dos tipos de requisitos, los requisitos de usuario y los de sistema. Los primeros recogen como interactúa el usuario con el sistema y los segundos cómo interactúan los distintos sistemas entre sí. Por tanto, el sistema, debe cumplir los siguientes requisitos:

- **Requisitos de usuario**:
  - *Requisito 1*: El sistema debe permitir introducir un texto en una ventana de texto o bien cargando un fichero de texto.
  - *Requisito 2*: El sistema debe permitir que el usuario seleccione el tipo de entidad o entidades a mostrar, pudiendo elegir entre PERSONA, ORGANIZACIÓN o LOCALIZACION.
  - *Requisito 3*: el usuario debe poder seleccionar las herramientas que analizarán del texto, pudiendo elegir una, dos o las tres.

- **Requisitos del sistema:**
  - *Requisito 4*: los recursos seleccionados deben tener una precisión y cobertura aceptable para etiquetado de textos en castellano.
  - *Requisito 5*: el sistema debe ser accesible a través de internet, por lo cual se implementará a modo de forma de servicio web.

- *Requisito 6*: el Servicio web debe estar compuesto por tres herramientas comerciales siendo una de ellas un API.

- *Requisito 7*: la aplicación Web será el encargado de enviar las peticiones con el texto a analizar siendo este el que lo devuelva etiquetado.

- *Requisito 8*: aplicación Web será el encargado recoger el texto anotado y traducirlo a lenguaje HTML.

- *Requisito 9*: El sistema debe mostrar el texto etiquetado según PERSON, ORGANIZATION o LOCATION.

- *Requisito 10*: El sistema debe ser integrable con otros sistemas y/o aplicaciones.

4.2. ARQUITECTURA

El servicio web desarrollado en este proyecto consta de tres partes, por un lado una aplicación de reconocimiento y clasificación de entidades nombradas, otro la construcción de un servicio web a partir de la aplicación y por último la construcción de una interfaz web que consuma el servicio web.

4.2.1. ARQUITECTURA SOFTWARE

La arquitectura de software es un conjunto de patrones que proporcionan el marco de referencia necesario para guiar la construcción de un software, permitiendo a los programadores, analistas y desarrolladores de software compartir una misma línea de trabajo y cubrir todos los objetivos y restricciones de la aplicación. Es considerada el nivel más alto en el diseño de la arquitectura de un sistema puesto que establecen la estructura, funcionamiento e interacción entre las partes del software. Las arquitecturas más comunes son:

- *Monolítica*. Donde el software se estructura en grupos funcionales muy acoplados.

- *Cliente-servidor*. Donde el software reparte su carga de cómputo en dos partes independientes pero sin reparto claro de funciones.
- *Arquitectura de tres niveles*. Especialización de la arquitectura cliente-servidor donde la carga se divide en tres partes con un reparto claro de funciones: una capa para la presentación (interfaz de usuario), otra para el cálculo (modelado el negocio) y otra para el almacenamiento.

Sin embargo, actualmente existe una tendencia a implantar arquitecturas de desarrollo que proporcionen una base tecnológica fiable, moderna y segura que sirva como pilar del desarrollo de nuevos sistemas de información.

En este ámbito se enmarca la arquitectura SOA, una arquitectura orientada a servicios donde las aplicaciones están formadas por servicios débilmente acoplados pero muy interoperables.

Dado que nuestro objetivo es la construcción de un Servicio Web con cierta funcionalidad y que debe estar disponible para ser utilizada por otros servicios, sistemas, etc., la arquitectura más adecuada era precisamente una arquitectura SOA. Este tipo de arquitecturas comunica unos sistemas con otros mediante mensajes independientemente del lenguaje de programación y de la arquitectura hardware que tenga cada uno. Por lo tanto, solo es necesario definir la interface de cada servicio. Como en la gran mayoría de arquitecturas SOA los mensajes de comunicación entre sistemas son mensajes XML que siguen el protocolo WSDL donde se indica la descripción del servicio en cuestión.

Los beneficios que puede obtener una aplicación con arquitectura SOA son:

- Mejora en los tiempos de realización de cambios en procesos.
- Facilidad para abordar modelos de negocios basados en colaboración con otros entes (socios, proveedores).
- Capacidad para reemplazar elementos de la capa aplicativa SOA sin interrumpir el funcionamiento de la aplicación.
- Facilidad para la integración de tecnologías no relacionadas entre sí.

La figura 4 muestra la arquitectura de la aplicación a muy alto nivel, sin entrar en detalles de cómo interactúan los elementos entre sí.

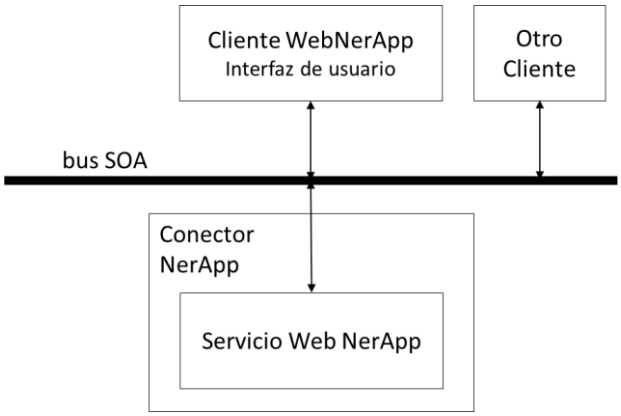


FIGURA 4: arquitectura SOA implementada.

4.2.2. INTERFACES

La comunicación entre los distintos componentes de la aplicación se realiza a través de las interfaces de cada componente. En la figura 5 se muestra la arquitectura software donde se puede ver la conexión entre los sistemas a través de las interfaces.

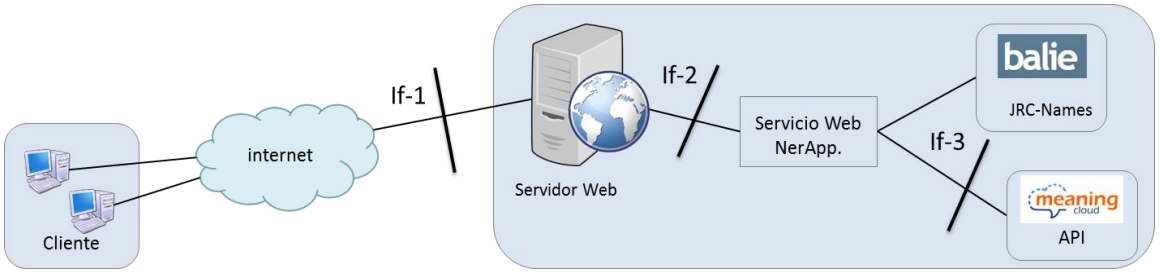


FIGURA 5: arquitectura software.

- Interfaz 1 (If-1): interfaz gráfica de la aplicación web.
- Interfaz 2 (If-2): conexión entre el aplicación web y el servicio web NerApp
- Interfaz 3 (If-3): conexión entre servicio web y API MeaningCloud.

4.2.2.1. Interfaz 1. Interfaz gráfica.

Es la interfaz gráfica del sistema. Desde aquí podrá hacer peticiones al Servicio Web desde cualquier navegadora través de una IP pública. Tiene como tarea recoger las peticiones del usuario y mostrarlo una vez analizado.

4.2.2.2. Interfaz 2. Conexión entre aplicación web y servicio web.

Este es el grueso de la aplicación. La aplicación web se encarga de enviar el texto recogido por la interfaz web al servicio de reconocimiento y clasificación de entidades. Este analiza el texto según los requisitos del usuario y lo envia de vuelta a la aplicación web que lo traducirá a HTML para que el navegador pueda interpretarlo.

4.2.2.3. Interfaz 3. Conexión entre servicio web y API.

Esta interfaz es la encargada de realizar la conexión entre nuestro servicio web y el API de MeaningCloud. Se encarga de envía el texto junto con los parámetros solicitados para el análisis del texto a través de HTTP.

4.3. DISEÑO E IMPLEMENTACION DE SISTEMA.

Cómo ya se vio en el apartado anterior, uno de los requisitos del sistema planteados era que el sistema construido debe facilitar la integración con otros sistemas o aplicaciones. Esto representa un reto a la hora de implementar un sistema dado que existen gran variedad de lenguajes y sistemas operativos disponibles para ello. Además, teniendo en cuenta que el acceso a Internet es permanente, la solución seleccionada fue la construcción de un servicio web, implementado bajo la arquitectura SOA, ya que un servicio web puede ser consumido independientemente del lenguaje de programación utilizado con simplemente configurar la capa de intercambio.

El servicio web de identificación y clasificación de entidades se puede dividir en tres partes. Por una parte están los recursos de reconocimiento y clasificación de entidades, otra parte es el Servicio Web que permite que el sistema sea consumible por otras aplicaciones y por último la interfaz gráfica web.

En la figura 6 se muestra un esquema general del sistema desarrollado.

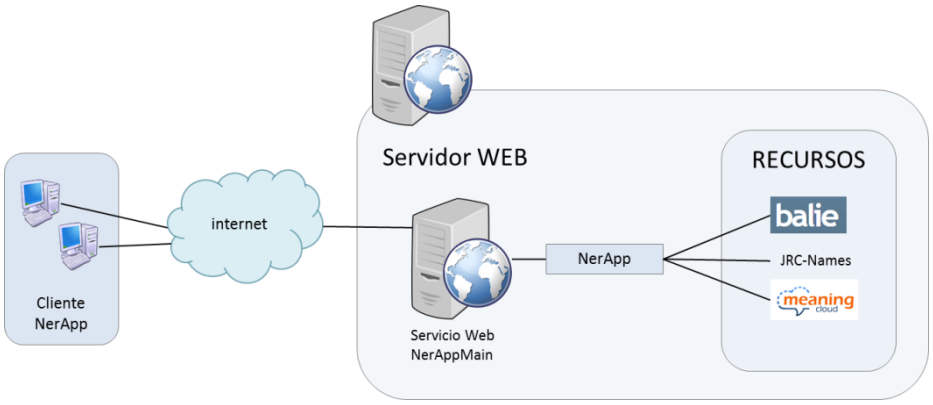


FIGURA 6: esquema del sistema.

- **Componentes del sistema e integración.**

El grueso del sistema lo llevan a cabo tres recursos de reconocimiento y clasificación de entidades encargados de las tareas de analizar y anotar el texto.

Cómo se ha mencionado en la introducción de esta sección, el sistema debe producir el texto anotado por uno, dos o los tres recursos. Cada una de las herramientas proporciona un etiquetado diferente, por lo que para la salida conjunta se ha definido un etiquetado común. La correspondencia de etiquetas se muestra en la tabla 2:

RECURSO	ETIQUETADO ORIGINAL	ETIQUETADO TRANSFORMADO

MeaningCloud	{
	"form": "Bin Laden",
	"id": "2cd8722f75",
	"sementity": {
	"class": "instance",
	"fiction": "nonfiction",
	"id": "ODENTITY_FULL_NAME",
	"type": "Top>Person>FullName"
	},
	"semId_list": [
	... //lista enlaces
	],
	"semtheme_list": [
	{
	"id": "ODTHEME_SECURITY_POLICE",
	"type": "Top>Society>Security Police"
	}
	],
	"variant_list": [
	{
	"form": "Osama Bin Laden",
	"inip": "114",
	"endp": "128"
	}
	],
	"relevance": "33"
	}
		[Osama Bin Laden @PERSON]

Balie	<ENAMEX TYPE="ORGANIZATION" ALIAS="5">Al Qaeda</ENAMEX>	**[Al Qaeda @ORGANIZATION]**
JRC-Names	found entity id = 1510 type pas Barack Obama (614)	**[Barack Obama @PERSON]**

TABLA 2: correspondencia de etiquetas.

Por ejemplo, dado el texto:

> *"El foco debe estar en matar y luchar contra los americanos y sus representantes. Es una de los notas que escribió Osama Bin Laden en una de las cartas desclasificadas por la Inteligencia de Estados Unidos entre los numerosos documentos incautados durante la operación de las fuerzas especiales que abatieron al jefe del grupo Al Qaeda en mayo de 2011.El portavoz de la Oficina de Inteligencia Nacional, Jeff Anchukaitis, ha explicado a la agencia AFP como la publicación de esta parte considerable de los documentos recuperados en la operación enlaza con el ejercicio de transparencia solicitado por el presidente Barack Obama."*

Debe devolver el texto como se muestra a continuación:

> *"El foco debe estar en matar y luchar contra los americanos y sus representantes. Es una de los notas que escribió* ***[Osama Bin Laden @PERSON]*** *en una de las cartas desclasificadas por la Inteligencia de Estados Unidos entre los numerosos documentos incautados durante la operación de las fuerzas especiales que abatieron al jefe del grupo* ***[Al Qaeda @ORGANIZATION]*** *en mayo de 2011.El portavoz de la Oficina de Inteligencia* ***[Nacional @ORGANIZATION], [Jeff Anchukaitis @PERSON],*** *ha explicado a la agencia* ***[AFP @ORGANIZATION]*** *como la publicación de esta parte considerable de los documentos recuperados en la operación enlaza con el ejercicio de transparencia solicitado por el* ***[presidente Barack Obama @PERSON]"***

La salida del sistema va a ser reutilizada por el cliente web desarrollado, es por ello que se han evitado los solapamientos de entidades en el etiquetado. Esto es debido a que al traducir el texto a lenguaje HTML entendible por el cliente web, si encuentra alguna entidad solapada produce un error.

Los recursos integrados en el sistema son *JRC-Names, Balie* y *MeaningCloud*. Los dos primeros son herramientas comerciales desarrolladas en java. El tercero es un API proporcionada por MeaningCloud.

MeaningCloud es un API, accesible a través de Internet, que permite a desarrolladores integrar funcionalidades de reconocimiento y clasificación de entidades a sus aplicaciones.

Para incluir este motor como parte del servicio, MeaningCloud proporciona dos clases desarrolladas en Java:

- *Post.java*: clase encargada de realizar las peticiones con los parámetros vía HTTP al API.
- *TopicsClient*. Java: esta clase es la encargada de setear los argumentos de entrada del API entre los que se encuentran: URL, clave de usuario, texto, entidades a anotar, idioma, salida, etc. enviarlos a la clase Post. Una vez el API devuelta el retorno, esta clase da formato a las entidades en forma de lista.

Las entidades y tipos que MeaningCloud ha encontrado en texto se recogen en el método principal TopicsCliente, así como la salida XML proporcionada de donde se extraen las posiciones inicial y final de cada una de las entidades en el texto.

La clasificación que MeaningCloud proporciona para cada entidad es según tipos y subtipos. Para este trabajo solamente nos quedamos con clasificación según tipos (PERSON, LOCATION, ORGANIZATION).

Por su parte, JRC-Names también proporciona unas clases y métodos desarrollados en Java para el etiquetado de textos sin necesidad de programación adicional. Solamente basta con acoplar las clases y librerías proporcionadas a nuestro proyecto. El paquete descargado de su página web contiene los siguientes archivos:

- *JRC-Names*.java.zip: es el paquete que contiene las clases y métodos que realizan la tarea de etiquetado de textos.
- *Entities.gzip*: es una lista de nombres y sus variantes. JRC_Names hace uso de este repositorio de nombres para hacer el maching con las entidades en el texto.
- *JRC-Names.jar*: es una librería que contiene el analizador y reconocedor de nombres propios y organizaciones.

En llamada a JRC-Names este recurso se ha de especificar la ruta donde está de la lista de nombres *Entities.gzi,* el texto a analizar y el idioma codificado en dos caracteres ("*es"* para español, "en" para inglés, etc.).

Una de las clases que proporciona este recurso trata las entidades con sus tipos y posiciones (inicial y final). Es en este método donde se recogen las entidades identificadas por JRC-Names y se añaden en las estructuras de almacenamiento definidas para guardar las entidades con su tipo y posiciones.

Por último, al igual que JRC-Names, Balie proporciona clases y métodos para reconocer y clasificar entidades. Entre la clases que forman el proyecto Balie, `NamedEntityRecognitionNerf` es la encargada de realizar la llamada a las demás para analizar el texto, pero necesita una clase por encima que inicialice y resetee algunos componentes como el tokenizador, las reglas de desambiguación o el lexicón. Esta clase se muestra en el cuadro 1.

```
public static String MakeBalie(String texto){

Tokenizer tokenizer = new Tokenizer(Balie.LANGUAGE_SPANISH, true);

        LexiconOnDiskI lexicon = new LexiconOnDisk(LexiconOnDisk.Lexicon.OPEN_SOURCE_LEXICON);
        DisambiguationRulesNerf disambiguationRules = DisambiguationRulesNerf.Load();

        tokenizer.Reset();
        tokenizer.Tokenize(texto);
        TokenList alTokenList = tokenizer.GetTokenList();

        NamedEntityRecognitionNerf ner = new NamedEntityRecognition-
Nerf(alTokenList,lexicon,disambiguationRules,new PriorCorrectionNerf(), NamedEntityTypeEnumMap-
pingNerf.values(),true);
        ner.RecognizeEntities();

        alTokenList = ner.GetTokenList();
String strAnnotated = alTokenList.TokenRangeText(0, alTokenList.Size(), false, true, true, true,
false, true);
return strAnnotated; }
```

CUADRO 1: Clase PruebaBalie.java.

Esta clase PruebaBalie.java recibe el texto y lo devuelta anotado al método que realizó la llamada. Dado que el texto retornado es en XML, se han de extraer las entidades y tipos. Balie no proporciona las posiciones inicial y final de cada entidad en el texto, por lo que para poder obtenerlas se han buscado las entidades en el texto origen y una vez tenemos la combinación entidad-tipo-posiciones se añaden a la estructura de almacenamiento donde también se han guardado las entidades detectadas por MeaninCloud y JRC-Names.

La comunicación entre el servicio web y los recursos descritos la llevan a cabo dos clases NerApp, encargada de proporcionar toda la funcionalidad al servicio web, y NerAppMain la clase sobre la que se ha creado el servicio web, es decir, la interface entre la aplicación web y el sistema de reconocimiento y clasificación de entidades nombradas.

En la figura 7 se muestra como se comunican las distintas clases y recursos del sistema implementado.

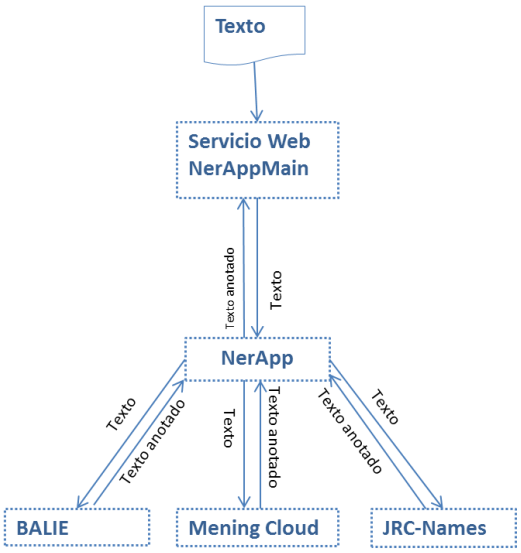


FIGURA 7: esquema de conexión entre recursos y servicio web.

NerAppMain recibe el texto y, dependiendo de los recursos que el usuario hay seleccionado, les envía el texto. NerApp recoge la salida anotada de cada recurso por separado, extrayendo las entidades, tipos y posiciones para guardarlos en MapTrees. Estas estructuras de almacenamiento java sirven para guardar datos en orden, así

conseguimos que las entidades se guarden en orden por la posición inicial. De este modo la tarea de marcar las entidades en el texto resulta más sencilla. Cómo hemos comentado al inicio de esta sección un ejemplo del "mapping" seleccionado es el siguiente *[Osama Bin Laden@PERSON]*. El método que edita las entidades en el texto se muestra en el cuadro 2.

```
public static String SalidaNerApp(String file) {

        ArrayList<Integer> temp  = new  ArrayList<Integer>();
                ArrayList<ArrayList<Integer>> posEnti  = new  ArrayList<ArrayList<Integer>>();

                int acum = 0;
        String cadena = file;

        Iterator <Integer> itEntidades = ordenado.keySet().iterator();
        while(itEntidades.hasNext()){
         temp = new ArrayList<Integer>();
         Integer id = itEntidades.next();
                 String entipo = ordenado.get(id);
                 String entidad = entipo.substring(entipo.indexOf(">")+1,entipo.indexOf("<"));
                 int poIni = Integer.parseInt(entipo.substring(0,entipo.indexOf(">")));
                 String tipo = entipo.substring(entipo.indexOf("<")+1,entipo.length());
                 cadena = cadena.substring(0,poIni+acum) + "[" + cadena.substring(poIni+acum , id + acum) +
" @" + tipo.toUpperCase() +"]" + cadena.substring(id + acum, cadena.length());
                 acum = acum + tipo.length() + 4;
        }
        return cadena;
    }
```

CUADRO 2: método encargado de marcar las entidades.

Como se ha comentado anteriormente, se han evitado solapamientos a la hora de anotar el texto. Para ello a la hora se insertar en las estructuras donde se han guardad las entidades se ha tenido en cuenta que:

- Si la entidad a insertar es parte de una entidad y además sus posiciones inicial y final son menores que la entidad ya existe en la estructura, se descarta.
- Si la posición inicial coincide con alguna de las entidades ya insertadas se descarta.
- Si la posición final coindice con alguna de las entidades ya insertadas, la entidad a insertar se descarta.

Además de esto, el orden de análisis establecido para los recursos es el siguiente e inserción en las listas es: MeaningCloud, JRC-Names, Balie.

El siguiente paso fue el desarrollo de una plataforma para dotar al sistema de un API web para que otras aplicaciones puedan hacer uso de este servicio. La clase NerAppMain desarrollada en Java es una capa que interactúa con los recursos y sobre la que está implementada servicio web.

Para implementar el servicio web se siguieron los pasos del asistente de Eclipse donde se definió como servidor Apache Tomcat y como motor de servicios web Apache Axis2. El Servicio web ha sido creado en modo Bottom-up, es decir, se crear la clase y a partir de esta el archivo WSDL donde está la descripción del servicio web entendible para otros servicios o aplicaciones sepan la funcionalidad de la que dispone, como se debe llamar, que parámetros espera y que datos devuelve

Los mensajes que intercambian la aplicación web, que ejerce de cliente, y el API son codificados en XML, y transportados a través del protocolo de transporte SOAP, de forma que el WSDL pueda interpretarlos.

El Servicio Web solamente dispone de una funcionalidad, por lo que el API se reduce a una llamada a la ruta http://localhost:8080/NerAppWS/services/NerAppMain usando un método POST de HTTP con el texto y los recursos seleccionados para el análisis como argumentos de entada. .

Para crear el cliente encargado de consumir el servicio web, se creó del mismo modo que el servicio web sobre la clase NerAppMain, a través del asistente de Eclipse. El asistente crear las clases necesarias para realizar la conexión:

- *CallbackHandler*: se encarga de la interacción con el usuario.
- *Stub*: contiene el código proxy para el cliente generado a partir del ficheros WSDL del servicio web.

El cliente implementado cuenta con dos clases, ClienteNerApp.java, encargada de recoger el texto introducido por el usuario en la interfaz gráfica y enviado al servicio web para que este lo analice y otra, *EtiquetadoHTML* que convierte el texto anotado a lenguaje legible para páginas HTML.

En el cuadro 3 se puede ver el contenido de ClienteNerApp.java encargada de realizar la conexión con el servicio web.

```
public static String Cliente (String[] args) {
        NerAppMainStub customer = null;
        NerAppMainStub.NerApp request = null;
        NerAppMainStub.NerAppResponse response = null;
        String respuesta = "";
try{
        //creamos el soporte y la petición
        customer = new NerAppMainStub();
        request = new NerAppMainStub.NerApp();

        // establecemos el parámetro de la invocación
        request.setArgs(args);

        // invocamos al web service
        try {
                response = customer.nerApp(request);
        } catch (NerAppMainExceptionException e) {

                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        // recogermos la respuesta
        respuesta = response.get return();

} catch (RemoteException excepcionDeInvocacion) {
        System.err.println(excepcionDeInvocacion.toString());
}
return(respuesta);
}
```

CUADRO 3: Clase ClienteNerApp.java.

Una vez definida la llamada al servicio, se desarrolla la Interfaz gráfica web para consumir el servicio web implementado. Se ha implementado como una aplicación web en JSP con HTML, CSS y JavaScript que le aporta funcionalidad.

Esta está compuesta por dos archivos JSP, uno encargado de las recoger el texto introducido por el usuario y sonde se pueden seleccionar las herramientas para el análisis, *webnerappRequest.jsp* y otra, muestra el texto etiquetado, *webnerappResponse.jsp*, y donde el usuario puede seleccionar las entidades que desea sean mostradas. La Interfaz, se ha diseñado teniendo en cuenta que la distribución de la información sea clara, este centrada e intuitiva. Tiene el siguiente aspecto (figura 8 y figura 9).

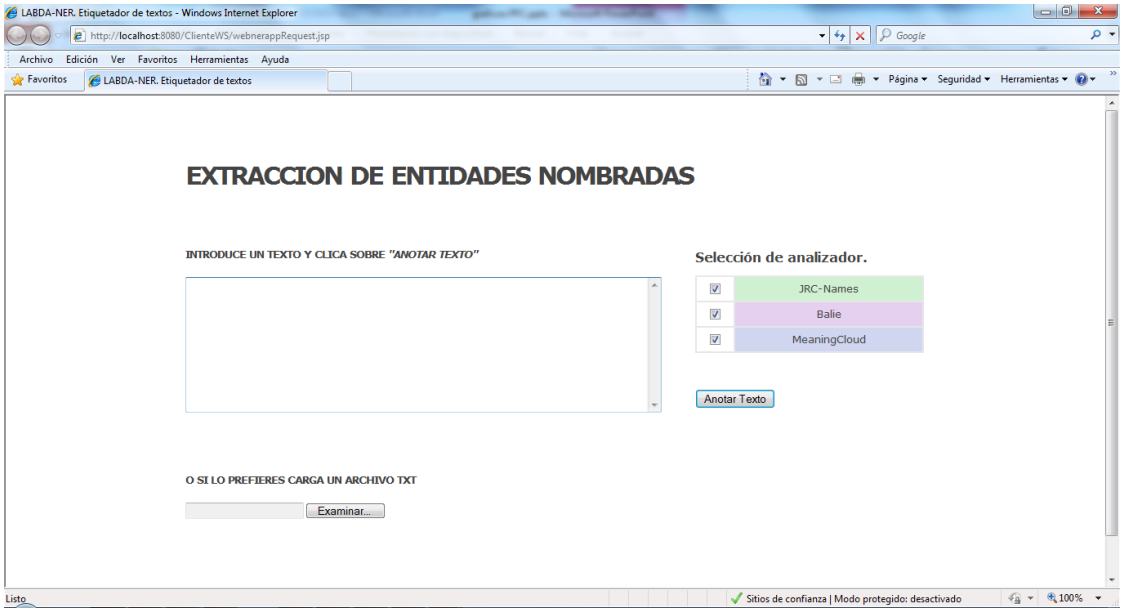


FIGURA 8: captura página web de inicio (webnerappRequest.jsp).

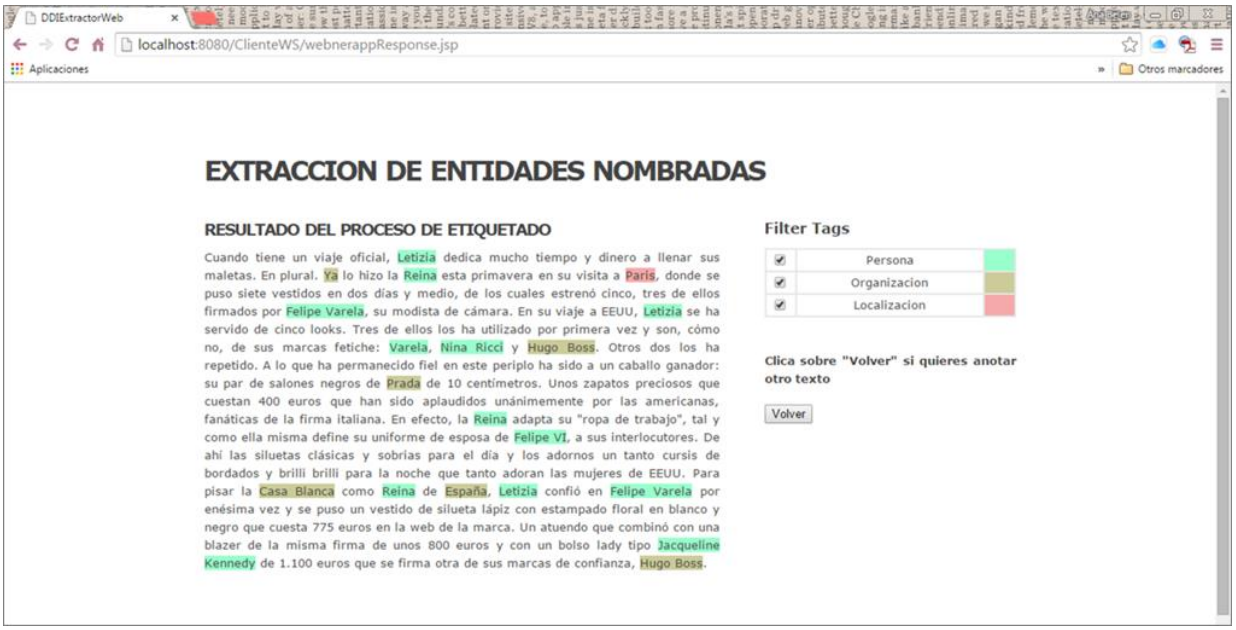


FIGURA 9: captura página web receptora del texto (webnerappResponse.jsp)

Como ya se ha mencionado, la implementación de la interfaz gráfica está enteramente desarrollada en HTML y para organizar los elementos dentro de la web, dar formato y color se han utilizado hojas de estilo CSS. Para aporta la funcionalidad

necesaria que requiere la selección de entidades a visualizar se ha utilizado JavaScript.

4.3.1 DIAGRAMA DE SECUENCIA

En el diagrama de secuencia representado en la figura 9 se muestra el paso de mensajes entre las partes del sistema.

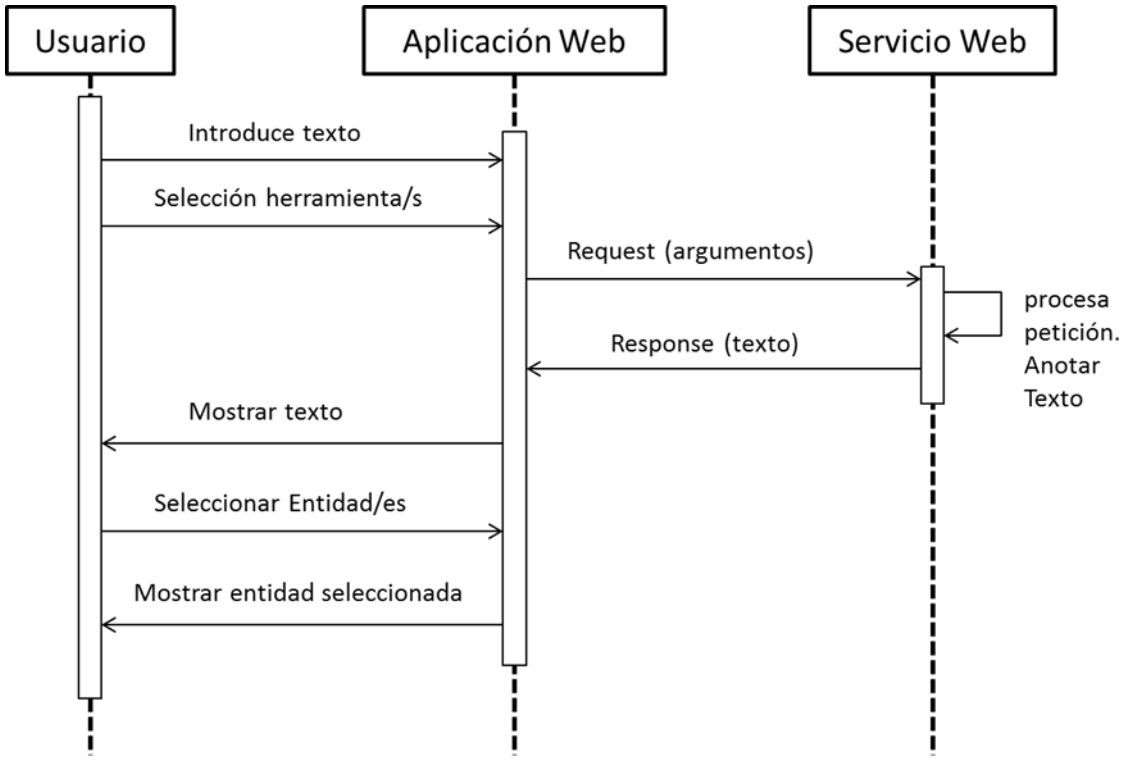


FIGURA 10: diagrama de secuencia.

4.3.3 ORGANIZACIÓN DEL PROYECTO.

En esta sección se representa la organización del proyecto a nivel de paquetes y clases (figura 10).

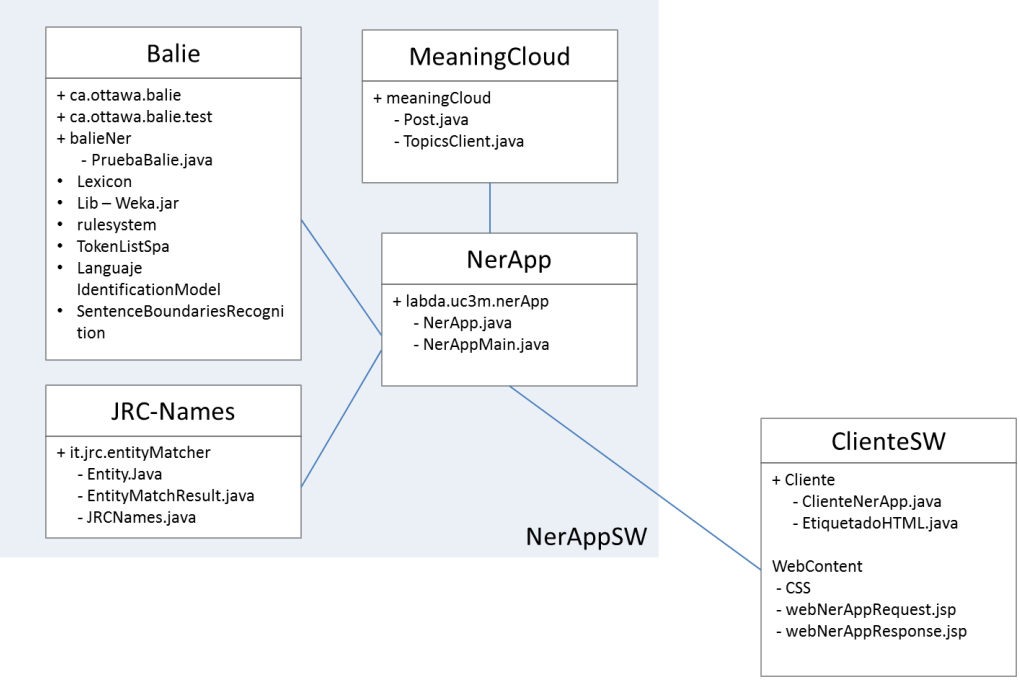


FIGURA 11: organización del proyecto.

CAPÍTULO 5.
PLAN DE PRUEBAS Y EVALUACIÓN

En este capítulo se evalúa la tarea de reconocimiento y clasificación de entidades nombradas (REN) de servicio web construido. REN es una subcategoría de la EI que busca localizar y clasificar los elementos del texto en categorías predefinidas, que vamos a centrar en PERSONA, LOCALIZACION y ORGANIZACIÓN.

Los sistemas de reconocimiento de entidades nombradas actuales han llegado a conseguir un rendimiento casi humano para el inglés. Por ejemplo en la conferencia MUC-7 [37] el mejor sistema consiguió una medida-F del 93,37% mientras que el reconocimiento por parte del humano obtuvo un 97,60%. A pesar de estas prometedoras cifras los sistemas REN desarrollados para un dominio no proporcionan tan buenos resultados en otros ámbitos, lo que da lugar a realizar grandes esfuerzos para adaptarlos. En la actualidad, la tarea principal está centrada en reducir la mano de obra mediante técnicas de aprendizaje semi-supervisado para conseguir mayor robustez entre dominios.

Este estudio se centra en evaluar el reconocimiento y clasificación de entidades nombradas que proporciona el servicio web compuesto por tres herramientas REN.

Este análisis se va a llevar a cabo teniendo en cuenta las variables precisión y exhaustividad encargadas de medir el rendimiento de los sistemas de búsqueda y recuperación y clasificación de información [38].

- *Precisión*: la fracción de instancias recuperadas que son relevantes.
- *Exhaustividad*: fracción de instancias relevantes que han sido recuperadas.
- *Valor-F*: media armónica de las dos medidas anteriores para comprobar cuán lejos se encuentran de la *utilidad teórica* (situación teórica en la cual existe una precisión y exhaustividad alta, es decir, cercana al 1).

Las fórmulas para calcular estas medidas son las siguientes:

$$\text{PRECISIÓN} = \frac{|\{\text{documentos relevantes}\} \cap \{\text{documentos recuperados}\}|}{|\{\text{documentos recuperados}\}|}$$

$$\text{EXHAUSTIVIDAD} = \frac{|\{\text{documentos relevantes}\} \cap \{\text{documentos recuperados}\}|}{|\{\text{documentos relevantes}\}|}$$

$$\text{VALOR-F} = 2 \text{ X} \frac{\text{Precisión x Exhaustividad}}{\text{Precisión + Exhaustividad}}$$

5.1 MUESTRA DE DATO

Para analizar el desempeño del sistema se utilizará la batería de textos en castellano proporcionada por el corpus CoNLL-2002 [40], desarrollado con el objetivo de evaluar sistemas NERC basados en aprendizaje automático, por ello que está compuesto de tres archivos: train, testa, testb. Dado que nuestro sistema está compuesto por tres herramientas basadas en listas y no necesitan entrenamiento, solamente vamos a utilizar el archivo testb ya que es el que contiene el corpus.

Para las anotaciones utiliza cuatro clases, peronas (PER), lugares (LOC), organizaciones (ORG) y otro tipo de entidades que clasifica como MISC. Nosotros solamente vamos a analizar la precisión y cobertura en cuanto a personas, lugares y organizaciones.

El corpus testb de CoNLL-2002 es una colección de textos de artículos de la agencia EFE que contiene 53.049 tokens y 3.558 entidades. Cada uno de los token se

corresponde con una línea y las NE vienen marcadas siguiendo el formato BIO. Con la etiqueta B – 'tipoEntidad' indica el inicio de una entidad, mientras que el resto de tokens perteneciente a la misma entidad se marcan con I- 'tipoEntidad'. Por último, un token marcado con la etiqueta O indica que no se considera entidad. Este tipo de etiquetado asume que las EN no son recursivas ni se superponen.

Para preparar los datos, se ha creado un conversor CoNLL-2002 – texto que deshace la tokenización y produce un fichero con una sola línea donde cada texto está separado por un guion.

Para probar el sistema, se ha creado un nuevo método que, con cada uno de los textos del corpus, llama al servicio web de reconocimiento y clasificación de entidades y guarda el resultado en una variable para, posteriormente, guardar todo el corpus anotado en un fichero de salida. Este fichero contiene todas las anotaciones de EN encontradas en el texto.

El fichero resultante contiene todas las entidades que el sistema desarrollado ha detectado. Además, se obtiene un fichero con el número de entidades por tipología detectadas. Cómo el etiquetado del corpus y el diseñado para el sistema difieren, se ha definido una correspondencia (tabla 3) de etiquetas para poder obtener el número de entidades correctas. En este sentido, se considera que la entidad es correcta si coincide en número de tokens que compone la entidad y el tipo.

ETIQUETA CoNLL02	**ETIQUETA NerApp**
[Ciaran Power@PERSON]	Ciaran B-PER Power I-PER
[Cuerpo Nacional de Policía@ORGANIZATION]	Cuerpo B-ORG Nacional I-ORG de I-ORG Policía I-ORG
Infanta B-PER Isabel I-PER	[Isabel@PERSON]

TABLA 3: correspondencia etiquetado CoNLL – NerApp.

En la tabla 3 se puede observar las entidades marcadas como correctas y las marcadas como incorrectas (sombreado rojo).

5.2 ANÁLISIS DE RESULTADOS

Tal como se ha mencionado anteriormente, se evaluará el rendimiento del sistema calculando la precisión, cobertura y su media armónica F. Los resultados obtenidos se muestran en la tabla 4.

TIPOLOGÍA	EN Totales	EN Encontradas	EN Correctas	P (%)	R(%)	F(%)
PER	735	922	647	70,17	88,03	78,09
LOC	1.084	1.180	721	61,10	66,51	63,69
ORG	1.400	1.237	676	54,65	48,29	51,27

TABLA 4: resultados de la evaluación con CoNLL – 2002.

Si nos fijamos en los resultados obtenidos, los mejores corresponden a la tipología PER. El grado de precisión es aceptable, identifica un 70,17% de las EN del corpus, siendo la cobertura del 88.03% lo que indica que la mayoría de las entidades detectadas son correctas.

Analizando las listas manualmente, reconocimientos de entidades de PER falla cuando los nombres están formados por iniciales, como por ejemplo M.S.D. Otro dato considerar es que como no se ha tenido en cuenta la tipología MISC algunas entidades clasificadas como MISC por CoNLL, el sistema desarrollado las etiqueta como PER si entre sus token aparece el nombre de una persona. Por ejemplo, la entidad "*Premio de Periodismo Miguel Delibes*" CoNLL la clasifica como MISC mientras que NerApp remarca "*Miguel Delibes*" como PER.

La precisión para la tipología LOC es ligeramente más baja que la obtenida para PER pero sigue siendo aceptable. En cuanto a la cobertura se obtiene un valor bastante aceptable con respecto a la precisión, pues más de la mitad de las entidades encontradas son correctas. En este sentido los fallos en el etiquetado encontrados en la inspección manual son clasificar una entidad LOC como parte de una ORG.

En cuanto a la clasificación de ORG los datos obtenidos son más bajos que para PER o LOC. Aun así la media armónica supone un 51,7%. Aquí es donde el sistema más falla a la hora de reconocer y clasificar las entidades de un texto. Son

bastante habituales los fallos en cuanto al número de tokens que forma la entidad. Por ejemplo, en la entidad "*Sociedad de Oncología Clínica"* NerApp solamente clasifica como ORG "*Sociedad"*, siendo el número de tokens que la componen incompleto para contarla como correcta.

Entramos ahora a analizar las entidades que han sido detectadas por el sistema desarrollado y no debería (falsos positivos) y entidades que no ha detectado y si debería (falsos negativos).

Encontramos que, en la clase PER, tal como hemos comentado antes, el sistema no detecta nombres con siglas del tipo "*M.S.S"*. Por otro lado, como falsos positivos encontramos tokens que aparecen por completo en mayúsculas, del tipo "*GUERRA", "OCUPADA"* y algunos tokens considerados artículos que identifica como parte de nombre propios, como por ejemplo "*Al"* (se puede considerar parte de un nombre árabe).

En cuanto a la clase ORG como falsos positivo tenemos entidades del tipo "*TELECOMUNICACIONES"* o "BOLSA", que puede considerar como parte de una organización aunque realmente se encuentran solas. En cuanto a entidades que no se han detectado y el sistema si debería, falsos negativos, encontramos más variedad, como por ejemplo "*EFECOM", "ASAJA"* o nombres de asociaciones o instituciones más largos como por ejemplo "*Asociación de la Prensa de Valladolid"* o *"Consejería de Asuntos Sociales"*. Esto puede ser debido a que las listas de nombres que forman parte de los recursos no contemplan estas asociaciones.

Por último nos fijamos en la clase LOC. Esta clase presenta resultados aceptables, pero aun así hay entidades que no ha reconocido y si lo son (falsos negativos), como por ejemplo acrónimos de países ("*CZE", "ITA", "HOL"*) o algunos nombres de poblaciones como "*Remolinos"* o *"Malpartida"* y localizaciones extranjeras ("*al-Khiam"*). En cuando a los falsos positivos encontramos menos casos, destacar que identifica Ginebra considerara bebida en el corpus y algún artículo, como "*De"*, que puede interpretar como un acrónimo.

Concluyendo, el sistema implementado obtiene unos resultados aceptables en general no tanto así para la clase ORG. Aun así teniendo en cuenta el análisis detallado por clases considero que, es una muy buena herramienta para la detección de nombres propios.

CAPÍTULO 6.
PLANIFICACIÓN Y PRESUPUESTO

En esta sección se detalla en primer lugar las fases del proyecto y su planificación para continuar con los costes del mismo a nivel de personal y material.

6.1 FASES DEL PROYECTO

1- **Análisis recursos y herramientas**: en esta fase se analizaron varias herramientas comerciales y entornos de desarrollo de aplicaciones PLN para seleccionar aquellas que mejor se adecuaran a los requisitos planteados (descritas en la sección 2 y 3 de este documento). La mayor dificultad de esta fase se encuentra en la comprensión las herramientas PLN así como construir un entorno de trabajo ideo para el desarrollo del proyecto.

2- **Desarrollo e integración de recursos:** es la parte que más tiempo ha ocupado en el desarrollo del proyecto. En primer lugar hubo preparar el entorno de trabajo, lo que conlleva comprender como funciona cada uno de los componentes software que instalados y que función desempeñarán. Comprender como funcionan las herramientas seleccionadas y de qué manera integrarlas en el conjunto del proyecto. A continuación se definió la capa encargada generar una única salida y diseñar el "mapping" común.

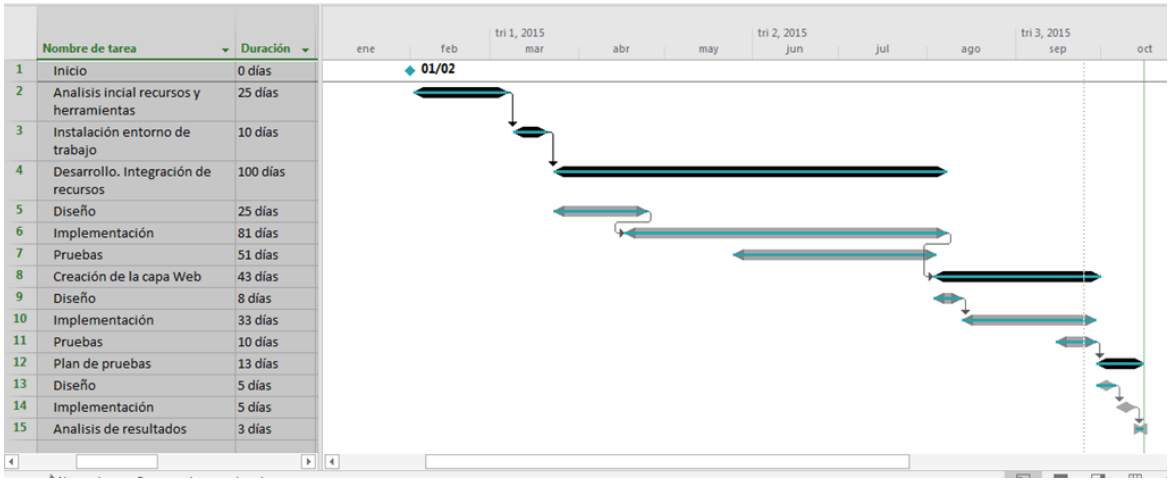
3- **Creación de la capa web:** es la última fase de desarrollo del proyecto. En ella se proporciona una interfaz web al sistema para interactuar con el usuario. La parte complicada fue la compresión de la implementación del servicio web, las capas de las que consta y como sería la comunicación entre los recursos y la interfaz web a través del servicio web.

4- **Plan de pruebas:** es la fase final del proyecto y en ella se comprueba la precisión y cobertura del sistema construido. La dificultad de esta etapa radica en el tratamiento del corpus en cuanto a la conversión del corpus en texto y viceversa.

Las fases 2,3 correspondientes al desarrollo y pruebas del proyecto constan a su vez de 4 fases:

- **Diseño:** donde se diseñan las interfaces, en caso que sea necesario, y la estructura interna. También se lleva a cabo un proceso de investigación, análisis y aprendizaje para conocer las tecnologías empleadas.
- **Implementación:** donde se desarrollan a nivel de código las tareas necesarias.
- **Pruebas:** para comprobar que todo funciona según los requisitos marcados
- **Documentación:** en esta fase se documenta tanto las decisiones tomadas así como el modelo de desarrollo de cada una de las tareas.

Teniendo en cuenta todo lo anterior además del desconocimiento de cada una de las herramientas, entornos de desarrollo, lenguajes de programación utilizados y la complejidad en este proyecto, se calcula el tiempo empleado en cada una de estas tareas. Todo el software utilizado es de licencia libre, por lo que no se va a incluir en el presupuesto. La tabla 10 representa el tiempo empleado en cada una de las tareas del proyecto.



CUADRO 4: Planificación del proyecto**.** Diagrama de Gantt.

6.2 PRESUPUESTO

Gastos de materiales (cuadro 3):

Concepto	Coste/unidad	Total
Ordenador de sobremesa	879,00	879,00
Servidor Web	2.500,00	2.500,00
Licencia Windows 7	300,00	300,00
	Total	**3.679 €**

Tabla 5: gastos de material.

En cuanto a los gastos de personal, el proyecto ha tenido una duración de 7 meses durante el cual se ha empleado una media de 4 horas diarias de lunes a viernes con un precio de 32€/h. No se han tenido en cuenta los festivos ni las vacaciones del personal. Se pueden ver en detalle en el cuadro 4:

Concepto	Horas	Total
Analisis recursos y herramientas	100	3.200,00
Desarrollo e integración de recursos sel	375	12.000,00
Creación de la capa web	150	4.800,00
Plan de pruebas	23	736,00
	Total	**20.736 €**

Tabla 6: gastos de personal.

Para el cálculo también se ha tenido en cuenta los gastos indirectos relacionados con el consumo de equipos, así como Internet y material de oficina. El cuadro 5 y 6 se puede ver en detalle el desglose:

Concepto				**Total**
Consumo de equipos	648 h	0,07 (kW)	0,124667 €/kWh	5,65
Cosumo luz	324 h	0,015 (kW)	0,124667 €/kWh	0,61
Potencia contratada (término fijo)	27 días	3,45 (kW)	0,115187 €/kWh	10,73
			Subtotal	16,99
			Impuesto eléctrico 5,11%	0,85
			Total	**17,8 €**

TABLA 7: gastos de indirectos.

Concepto			**Total**
Internet - ADSL	7 meses	30 €/mes	210,00
Material de oficina	-	-	20,00
		Total	**52,7 €**

TABLA 8: otros gastos indirectos.

El cuadro 7 que se muestra más abajo resume el presupuesto final del proyecto sumando el coste de material y personal.

Concepto	**Total**
Gastos de material	3.679,00
Gastos de personal	20.736,00
Gastos indirectos	17,8 €
Internet - ADSL	210,0 €
Material de oficina	20,0 €
Subtotal	**24.662,84**
IVA (21%)	5.179,20
Beneficios (15%)	3.699,43
TOTAL	**33.541,46 €**

TABLA 9: presupuesto total.

CAPÍTULO 7. CONCLUSIONES

Una vez concluido el proyecto, considero importante comprobar si se han cumplido los objetivos que se propusieron en su planteamiento.

Los pilares de este proyecto eran dos, el primero consistía en la construcción de un sistema de reconocimiento y clasificación de entidades nombradas a partir de tres recursos y como segunda parte crear un servicio web a partir del sistema desarrollado de manera que pudiera ser accesible desde cualquier aplicación. Además decidimos crear una aplicación web, donde se incluyeran opciones de selección de herramienta de análisis o entidades, que utilizara el servicio web.

Cómo objetivo principal y el motivo por el cual se desarrolla este proyecto es comprobar si al analizar un texto con tres recursos los resultados obtenidos son mejores que si solamente se hubiera analizado con una herramienta, es decir, alcanzar una mayor precisión y cobertura. Desde mi punto de vista, con la clase PER se ha conseguido, ya que llegar al 88% de cobertura es un valor bastante aceptable, no siendo así para la clase LOC y menos para la clase ORG. En este sentido, dado que los sistemas están basados en listas, si éstas son actualizadas con cierta regularidad se podría conseguir mejores resultados que los obtenidos.

En este sentido, se puede modificar el convenio de inserciones en las estructuras de almacenamiento o el orden en el que el texto es analizado por los recursos. Esto puede proporcionar mejoras en alguna de las clases, pero pérdidas en otras. Esto

podría tratarse como un trabajo futuro, analizar cualquier combinación posible con el fin de obtener mejores resultados de precisión y cobertura.

También surgen varios trabajos de ampliación a partir de este proyecto. Con los recursos que ahora mismo están integrados se puede ampliar el número de tipologías, ya que, por ejemplo, MeaninCloud proporciona más clases de tipos y subtipo. O también se pueden añadir más sistemas, herramientas o APIs de reconocimiento y clasificación.

GLOSARIO DE TÉRMINOS

- **PLN:** Procesamiento Natural del Lenguaje.

- **EI:** Extracción de Información.

- **EEN:** Extracción de Entidades Nombradas.

- **NER:** Reconocedor de Entidades Nombradas.

- **NEC:** Clasificación de entidades nombradas.

- **Entidad nombrada (EN):** Cadenas de palabras que representan nombres propios, organizaciones, lugares, fechas, etc.

    - **Persona (Per):** Limitadas a identificar Humanos por nombre, apellido o alias.
    - **Organización (ORG):** Limitadas a corporaciones, instituciones, agencias de gobierno, y demás grupos con una organización establecida.
    - **Lugar (LOC): Incluyen** nombres de lugares definidos política o geográficamente (ciudades, provincia, comunidad, país, región internacional, conjuntos de agua, montañas, etc.)y estructuras hechas por el hombre (Aeropuertos, autopistas calles, fábricas, monumentos.).

- **Gazetteer:** consiste en un conjunto de listas que contiene nombres de entidades como ciudades, organizaciones, días de la semana, etc. Estas listas son usadas para encontrar ocurrencias de esos nombres en el texto.

- **Token:** es una cadena de caracteres que tiene un significado coherente. Por ejemplo, en la frase "hola mundo", hola y mundo serían los tokens.

- **Tokenizar:** es el proceso de ruptura de un flujo de texto en elementos significativos llamad "Tokens". Es de utilidad tanto en lingüística como en ciencias de la computación donde forma parte del análisis léxico.

- **Trigger Words:** son un conjunto de palabras que dan información sobre el contexto para facilitar la clasificación de palabras.

- **XML:** del inglés *eXtensible Markup Language* (Lenguaje de marcas extensible) y es un metalenguaje extensible de etiquetas desarrollado por *W3C* (World Wide Web Consortium). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos, por lo tanto no es en sí un lenguaje, sino una manera de definir lenguajes para diferentes necesidades.

- **Parser:** se traduce como 'analizador sintáctico' y es una parte del compilar que transforma su entrada en un árbol de derivación, capturando la jerarquía implícita de la entrada. El analizador léxico crea los Tokens de entrada y estos son procesados por el analizador sintáctico para construir la estructura de datos (árbol de análisis o árboles de sintaxis abstracta).

- **Serializar:** Es el proceso de convertir un objeto en una secuencia de bytes para conservarlo en memoria, BBDD o archivo. La finalidad es guardar el estado del objeto para poder crearlo de nuevo cuando sea necesario.

- **Corpus lingüístico:** Es un extenso conjunto de ejemplos de uso de una lengua. Estos pueden ser textos o muestras orales.

- **URL**: Uniform Resource Locator. Es una cadena de caracteres con la cual se asigna una dirección única a cada uno de los recursos de información disponibles en Internet.

- **API: s**e define como Application Programming Interface (Interfaz de programación de aplicaciones) y es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una cama de abstracción.

- **Servlet, JSP**: módulos en Java que se ejecutan en un servidor HTTP (servidor Web) orientado a petición-respuesta.

- **Script:** es un programa simple que se suele alojar en un archivo de texto plano. Su uso es habitual en tareas de combinación de componentes, interacción con el sistema operativo o usuario.

- **JRE:** Java Runtime Environment. Necesario para ejecutar aplicaciones desarrolladas en Java.

- **JVM:** Java Virtual Machine. Necesario para poder ejecutar aplicaciones Java.

- **HTML:** HyperText Markup Language. Lenguaje de programación para elaborar páginas web.

- **CSS:** Hojas de estilo en cascada. Se utiliza en combinación con HTML para dar formato a las páginas web.

- **IDE:** Integrated development environmen. Es una aplicación que proporciona servicios integrales para facilitar el desarrollo de software. Habitualmente consiste en un etidor de código fuente, herramientas de construcción automáticas y un depurador de código.

- **JDK:** Java Development Kit. Es un software que proporciona herramientas de desarrollo para crear software. Entre sus archivos se encuentra el compilador de java javac.exe

- **Arquitectura SOA:** Arquitectura Orientada a Servicio

- **WSDL:** Web Services Description Languaje. Es un formato XML que se utiliza para describir servicios web

REFERENCIAS

[1] Evaluación *MUC . Disponible en esta URL: http://www-nlpir.nist.gov/related_projects/muc/index.html*

[2] **Rau, L.F.,** *Extracting Company Names from Text*. 1991

[3] Raquel Toribio Sardón. *Evaluación de la extracción de entidades nombradas de OpenCalais en textos no estructurados en castellano*. 2009/2010.

[4] Cynthia Costales Llerandi, José Hernández Palancar. *Estado del arte de la extracción de entidades nombradas*. Serie Gris. Minería de Datos. Versión digital. RNPS_ 2143 ISSN 2072-6260. Marzo 2010

[5] Técnicas estadísticas para el procesamiento del lenguaje natural. Universidad de Sevilla Víctor J. Díaz Madrigal

[6] Sekine, S., Named Entity: History and Future 2004.

[7] Cucerzan, S. and D. Yarowsky, Language Independent Named Entity Recognition Combining Morphological and Contextual Evidence. 1999.

[8] Collins, M. and Y. Singer, Unsupervised Models for Named Entity Classification. 1999.

[9] Nadeau, D., P. Turney, and S. Matwin, Unsupervised Named-Entity Recognition: Generating Gazetteers and Resolving Ambiguity. 2006.

[10] Mikheev, A., C. Grover, and M. Moens, Description of the LTG system used for MUC-7. 1998.

[11] Borthwick, A. A Maximum Entropy approach to Named Entity Recognition. 1999

[12] Srihari, R. and C.N.a.W. Li, A Hybrid Approach for Named Entity and Sub-Type Tagging. 2001.

[13] Cruz, Y.R., et al., Un enfoque híbrido al Reconocimiento de Nombres de Entidades para el español. 2007.

[14] Rob DiCiuccio. Entity extraction & Content API Evaluation. ViewChange.org ath the 2010 Global Philanthropy Forum. Available in http://blog.viewchange.org/2010/05/entity-extraction-content-api-evaluation

[15] **Zemanta**: http://www.zemanta.com/

[16] **OpenCalais**: http://www.opencalais.com/

[17] **ApiAlchemy**: http://www.alchemyapi.com/

[18] **Yahoo**: http://developer.yahoo.com/search/content/V1/termExtraction.html

[19] **OpenAmplify**: http://www.openamplify.com/

[29] **MeaningCloud**: https://www.meaningcloud.com/

[21] **JRC-Namnes**: http://langtech.jrc.it/JRC-Names.html

[22] **TextRazor**: http://www.textrazor.com/

[23] **UIMA**: http://uima.apache.org/

[24] **Balie**: http://balie.sourceforge.net/

[25] **YooName**: http://infoglutton.com/yooname-named-entity-recognition.html

[26] **GATE**: http://gate.ac.uk/

[27] **BRAT**: http://www.nactem.ac.uk/brat/static/case-studies.html

[28] **Eclipse**: http://www.eclipse.org/home/index.php

[29] **JAVA**: https://www.java.com/es/

[30] **Apache Tomcat**: https://tomcat.apache.org/download-70.cgi

[31] **Apache Axis2**: http://axis.apache.org/

[32] **Lenguaje JAVA**: https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

[33] **JSP**: https://es.wikipedia.org/wiki/JavaServer_Pages

[34] **JavaScript**: https://es.wikipedia.org/wiki/JavaScript

[35] **HTML:** https://es.wikipedia.org/wiki/HTML

[36] **CSS:** https://librosweb.es/libro/css/capitulo_1.html

[37] **MUC-7:** http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_toc.html#multiple

[39] **Precisión y cobertura**: https://en.wikipedia.org/wiki/Precision_and_recall

[40] http://www.aclweb.org/anthology/C96-1079

[41] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.27.5340

[42] http://www.cs.nyu.edu/~sekine/papers/NEsurvey200402.pdf

[43] CoNLL: Conference on Natural Language (Sang, 2002)(Sang & Meulder, 2003). http://dl.acm.org/citation.cfm?id=1119195

[44] https://www.ldc.upenn.edu/sites/www.ldc.upenn.edu/files/lrec2004-ace-program.pdf

[45] B. Settles. Active learning literature survey. Technical Report 1648, University of Wisconsin-Madison, 2009.

[46] https://gate.ac.uk/sale/ell2/ie/preprint.pdf

Evaluación de la extracción de Entidades Nombradas de Open Calais en castellano: http://journal.sepln.org/index.php/pln/article/view/811

APENDICE A.

TECNOLOGÍAS UTILIZADAS

En esta sección se hace mención sobre el software y lenguajes utilizados para la implementación del proyecto y que deben instalarse para el correcto funcionamiento del Servicio Web.

1. SOFTWARE

- **Eclipse JEE Juno SR2. IDE para desarrollo de software** [28]

Eclipse es una plataforma de desarrollo concebida desde sus orígenes para convertirse en una plataforma de integración de herramientas de desarrollo. No tiene un lenguaje específico, sino que es un IDE genérico, aunque goza popularidad entre la comunidad de desarrolladores del lenguaje Java usando el plug-in JDT que viene incluido en la distribución estándar del IDE y que nos proporciona información sobre la clase, método, palabra reservada, etc con solo poner el cursor encima. Proporciona herramientas para la gestión de espacios de trabajo, escribir, desplegar, ejecutar y depurar aplicaciones.

Como características de Eclipse destacan:

- Las distintas perspectivas, editores y vistas que permiten trabajar de forma óptima en un determinado entorno de trabajo
- Está basado proyectos. El IDE proporciona asistentes para la creación de proyectos, que no son más que un conjunto de recursos relacionados entre sí, como pueden ser el código fuente, documentación, ficheros de configuración, árbol de directorios, etc.,
- Incluye un potente depurador de código muy fácil e intuitivo y que visualmente ayuda a mejorar el código.
- Cuenta con una extensa colección de plug-ins disponibles a través del marketplace de Eclipse

- **Java JRE 8** [29]

Es necesario para poder ejecutar aplicaciones desarrolladas en Java dado que actúa como intermediario entre el sistema operativo y Java. Está formado por una máquina virtual de Java (JVM), un conjunto de bibliotecas Java y componentes necesarios para que una aplicación desarrollada en Java pueda ser ejecutada.

- **Servidor Web Apache Tomcat 7**. [30]

Tomcat es un contenedor web con siporte de servlets y Java Server Pages (JSPs), por lo que no es un servidor de aplicaciones. En su distribución, incluye un compilador Jasper que compila las JSP convirtiéndolas en servlets. Apache Tomcat fue desarrollado en Java, por lo que puede ejecutarse sobre cualquier sistema operativo previa instalación de la JVM, aunque también puede ejecutarse sobre cualquier otro sistema operativo. Además puede funcionar como servidor Web en entornos con alto nivel de tráfico y alta disponibilidad.

- **Motor de servicios web Apache Axis2.** [31]

Axis2 es un proyecto de la suite de Apache, basado en Java donde se implementa un cliente y un servidor de Servicios Web para SOAP, permitiendo mejorar la

portabilidad de los mismos. En general Axis provee un completo modelo de objetos basado en una arquitectura modular que facilita su funcionalidad y soporte para los servicios web. Permite entre otras cosas, el envío de mensajes SOAP (con o sin adjuntos), recepción y procesamiento de mensajes, creación de Servicios basados en clases Java, implementación de clases para Servidores y Clientes usando el WSDL, soporte a protocolos de Servicios Web (SOAP, REST) etc.

2. LENGUAJES DE PROGRAMACIÓN

- **JAVA.** [32]

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. A día de hoy es considerado una de los lenguajes de programación más populares, concretamente para aplicaciones cliente-servidor.

Es un tipo de lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. La finalidad es que una vez un programa haya sido escrito se pueda ejecutar en cualquier dispositivo (en inglés WORA, "write once, run anywhere).

- **JSP.** [33]

Es una tecnología que ayuda a los desarrolladores de software a crear páginas web dinámicas basadas en HTML, XML, entre otros tipos de documentos. Para desplegar y correr JPS, se requiere un servidor web compatible con contenedores servlet como Apache Tomcat y que la máquina tenga instalado el JVM.

El rendimiento de una página JSP es el mismo que tendría el servlet equivalente, ya que el código es compilado como cualquier otra clase Java. La principal ventaja de JSP frente a otros lenguajes es que el lenguaje Java es un lenguaje de propósito general que excede el mundo web y que es apto para crear clases que

manejen lógica de negocio y acceso a datos de una manera prolija. Esto permite separar en niveles las aplicaciones web, dejando la parte encargada de generar el documento HTML en el archivo JSP.

Otra ventaja es que JSP hereda la portabilidad de Java, y es posible ejecutar las aplicaciones en múltiples plataformas sin cambios.

- **JavaScript.** [34]

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

- **HTML.** [35]

Es un estándar que sirve de referencia para la elaboración de páginas web en sus diferentes versiones. Define una estructura básica y un código para la definición de contenido de una página web, como texto, imágenes, videos, entre otros. Es un estándar a cargo de la W3C, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la World Wide Web. Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.1

El lenguaje HTML basa su filosofía de desarrollo en la diferenciación. Para añadir un elemento externo a la página, no se incrusta directamente en el código, sino que se hace una referencia a la ubicación de dicho elemento. De este modo, la página web contiene sólo texto mientras que recae en el navegador web la tarea de unir todos los elementos y visualizar la página final.

- **CSS.** [36]

CSS es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML, XML y XHTML. Es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo. Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.

Al crear una página web, se utiliza en primer lugar el lenguaje HTML/XHTML para ***marcar*** los contenidos, para después utilizar el lenguaje CSS en la definición del aspecto de cada uno de los elementos.

Una de las características más importantes del lenguaje CSS es su modelo de cajas, que hace que todos los elementos de una página web se representen mediante cajas con diferentes capas. Las cajas de una página se crean automáticamente. Cada vez que se inserta una etiqueta HTML, se crea una nueva caja rectangular que encierra los contenidos de ese elemento. La figura 10 muestra el modelo de cajas de CSS.

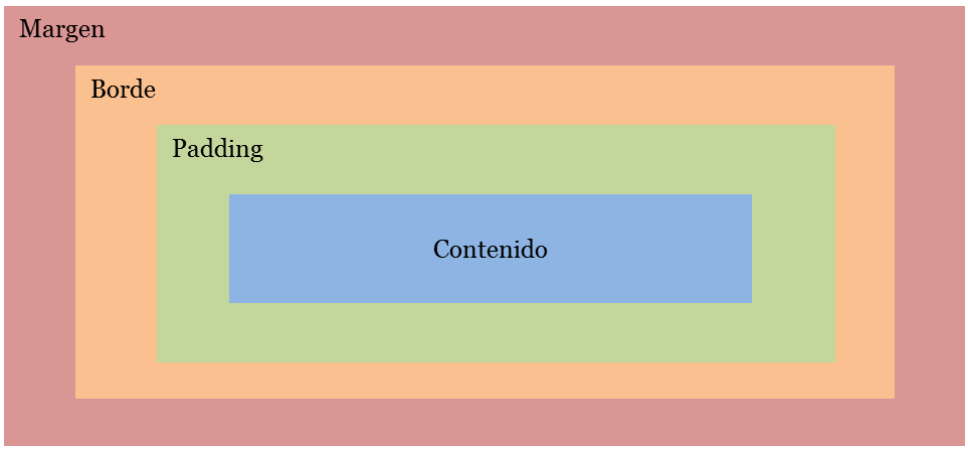


FIGURA 12: modelo CSS de cajas.

APENDICE B
PREPARACIÓN DEL ENTORNO DE TRABAJO

Como paso previo a la implementación del sistema desarrollado, hemos de preparar el PC.

El primer paso es instalar un **IDE** (Integrated Development Environment) con el cual desarrollar aplicaciones, tanto de escritorio como web. En nuestro caso hemos seleccionado **Eclipse Juno**. Para ello accedemos a la página de Eclipse https://eclipse.org/downloads/packages/release/juno/sr2 y descargamos la versión Java EE Developers (Figura 11) y la descomprimimos la carpeta ECLIPSE que crearemos en Archivos de Programa.

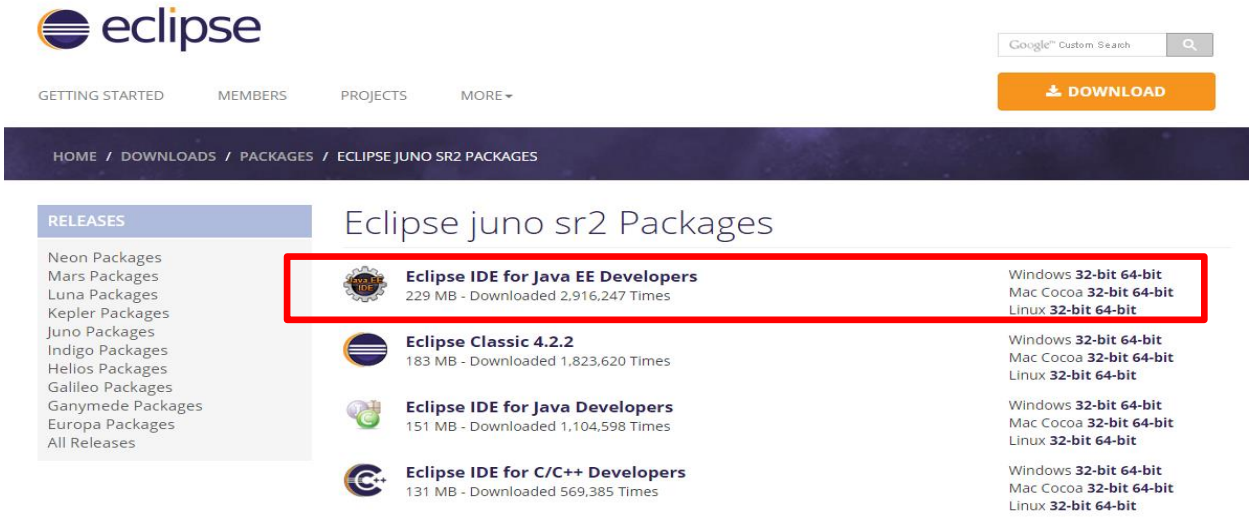


FIGURA 13: página de descarga de eclipse.

Cómo requisitos del sistema previos para que las aplicaciones desarrolladas Eclipse funcionen correctamente necesitaremos tener instalado en nuestro ordenador el JRE, el cual incluye la máquina virtual de java (JVM) y el JDK. El JDK (Java Development Kit) es un entorno de desarrollo para hacer aplicaciones, applets y varios componentes utilizando Java. Para ello accedemos a la web de Java, https://java.com/es/download/, y descargamos la última versión. Para instalarlos, crearemos la carpeta *C:\Archivos de programa\JAVA* e indicaremos al instalador que queremos que los descomprima ahí.

Una vez tenemos el entorno de Eclipse preparado, para ejecutarlo basta con hacer doble clic sobre el archivo **eclipse.exe** que se encuentra en la capeta donde se han descomprimido los archivos.

Se puede dar el caso que al ejecutar Eclipse muestre un error. Esto es debido a que las variables de entorno no están configuradas, por lo deberemos configurarlas para que el JRE y JDK estén vinculados al IDE. Para ello, accedemos a las variables de entorno, clicamos sobre el icono Mi PC con el botón derecho seleccionando *propiedades*. Dentro de esta ventana elegimos *configuración avanzada del sistema>opciones avanzadas* y accedemos a las variables clicando sobre "variables de entorno". En esta ventana pulsar sobre el botón "Nueva...". Una vez ahí, en el recuadro de Nueva variable del sistema introducimos lo siguiente (figura 12):

- Nombre de la variable: **JAVA_HOME.**
- Valor de la variable: **Ruta de donde está instalado el JDK de java**.

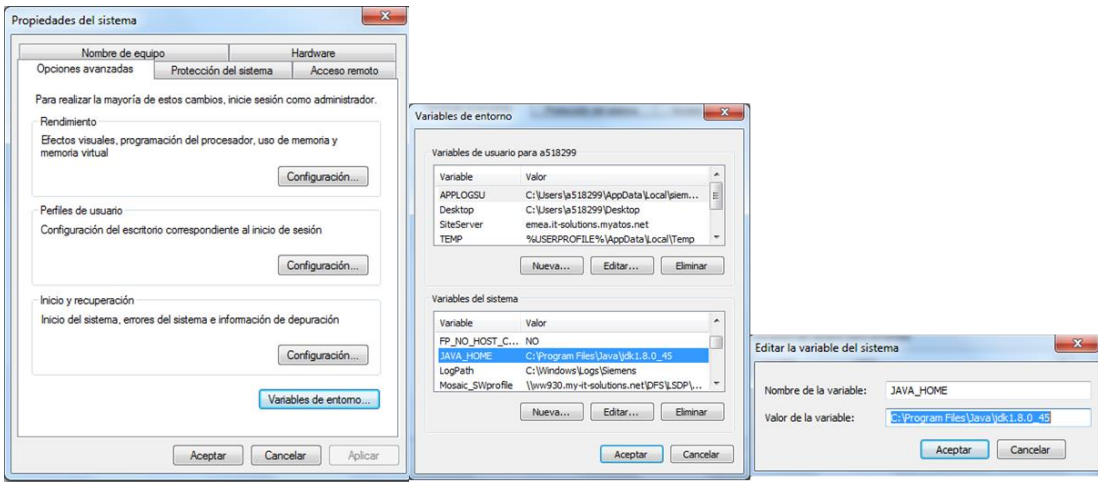


FIGURA 14: acceso y configuración de las variables de entorno.

También hay que configurar las variables CLASSPATH y path. Para ello seguimos los siguientes pasos:

- Buscar la variable CLASPATH, clicar sobre editar. Moverse al final del campo valor de la variable y añadir *";%JAVA_HOME%\lib\tools.jar;"*.
- Lo mismo hay que hacer con la variable path, editarla y añadir *";%JAVA_HOME%\bin"*.

Después de esto, Eclipse ya está configurado correctamente. A continuación definiremos el servidor web que se vamos a utilizar. En este caso hemos seleccionado Apache Tomcat 7. Para instalarlo en Eclipse:

- Descargar Apache Tomcat7 y descomprimir el archivo .zip en *C:\Program Files\Apache Software Foundation*
- A continuación hay que configurar las variables de entorno (figura 13). En este caso la variable CATALINA_HOME, a la cual se le debe asignar la ruta de instalación de Tomcat. Añadimos también a la variable CLASPATH la *libreria lib* de Tomcat y a la variable path la *libreria bin* del mismo modo que se ha hecho a la hora de configurar Eclipse.

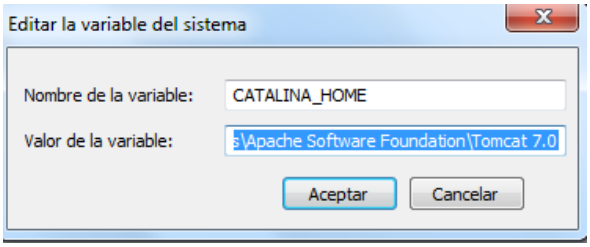


FIGURA 15: configuración variables de entorno para Apache Tomcat.

Para comprobar que el servidor se ha instalado correctamente es funciona, accedemos el *Command Prompt* de Windows y escribimos *%CATALINA_HOME%\bin\startup.bat* y pulsamos a *Intro*. Si se ha instalado correctamente nos aparecerá una ventana similar a la representada en la figura 14:

FIGURA 16: ejemplo de arranque del servidor**.**

Ahora definiremos Apache Tomcat como servidor para Eclipse. A continuación se detallan los pasos que se deben seguir.

Arrancamos Eclipse y en la barra de menú vamos a *Windows>Preferences (figura 15)*.

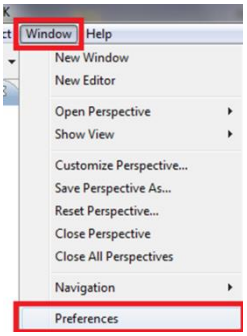


FIGURA 17: configuración Apache Tomcat en Eclipse.

Nos situamos en la columna de la izquierda de la ventana, clicamos sobre *Server* y dentro de este seleccionar *Runtime Environments,* tal como muestra la figura 16. Para añadir un servidor, clicamos sobre el botón "*Add…*" situado en la parte superior derecha. Se abrirá otra ventana más, en la que debemos seleccionar la carpeta de Apache Tomcat. En esta carpeta seleccionamos la versión de Tomcat instalada, en nuestro caso Tomcat7. Clicar sobre el botón ***Next*** para continuar.

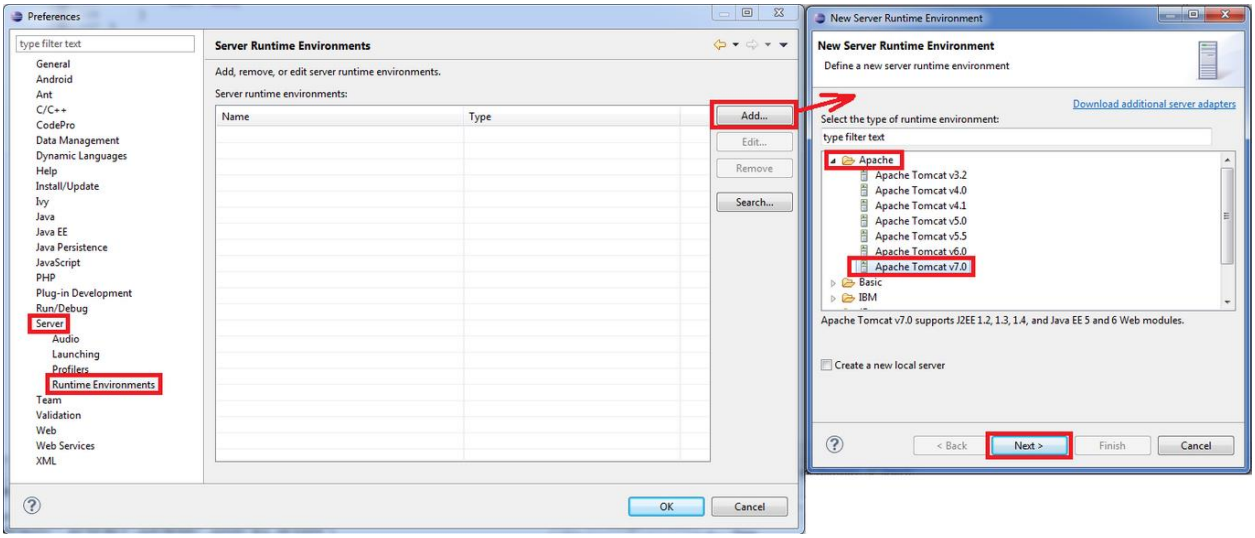


FIGURA 18: configuración Apache Tomcat en Eclipse II.

En la siguiente ventana buscar en *Tomcat installation directory* la carpeta donde se ha instalado Apache Tomcat en el PC y clicar en *Finish*. Un punto a tener en cuenta es que si se quiere que Apache Tomcat ejecute JSPs debe tener acceso no sólo a un JRE, sino a un JDK. Dado que nosotros si necesitamos ejecutar páginas JSPs en lugar de seleccionar de la lista JRE el JRE que tenemos instalado, seleccionamos el JDK tal como se ve en la figura 17.

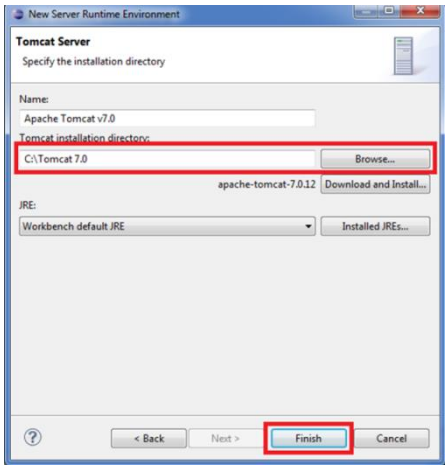


FIGURA 19: configuración Apache Tomcat en Eclipse III.

Como último paso, solamente queda configurar Eclipse para que utilice Apache Axis2 como motor de servicios web. **Apache Axis2** es una implementación

OpenSource de SOAP que proporciona un entorno de ejecución para servicios web implementados en Java y un compilador de Java a WSDL y de WSDL a Java.

Para ello, accedemos a la página de Apache Axis2 http://axis.apache.org/axis2/java/core/download.cgi y descargamos la distribución binaria y la WAR. A continuación creamos una carpeta para cada distribución en **C:\Archivos de programa** y descomprimimos cada versión donde corresponda.

Para vincular Tomcat7 y Axis2 copiamos el archivo **.WAR** en la carpeta *webapp* de la instalación de Tomcat7. Por ejemplo, si la instalación de Tomcat está en archivos de programas, la ruta será **C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps.**

Ahora indicamos a Eclipse que queremos a Apache Axis2 como motor de servicios web (figura 18). Para ello en la opción **Web Services>Axis2 Preferences** indicamos la ruta donde se ha descomprimido la distribución binaria de Apache **Axis2** y clicamos en finalizar. Y con esto el entorno de trabajo está preparado.

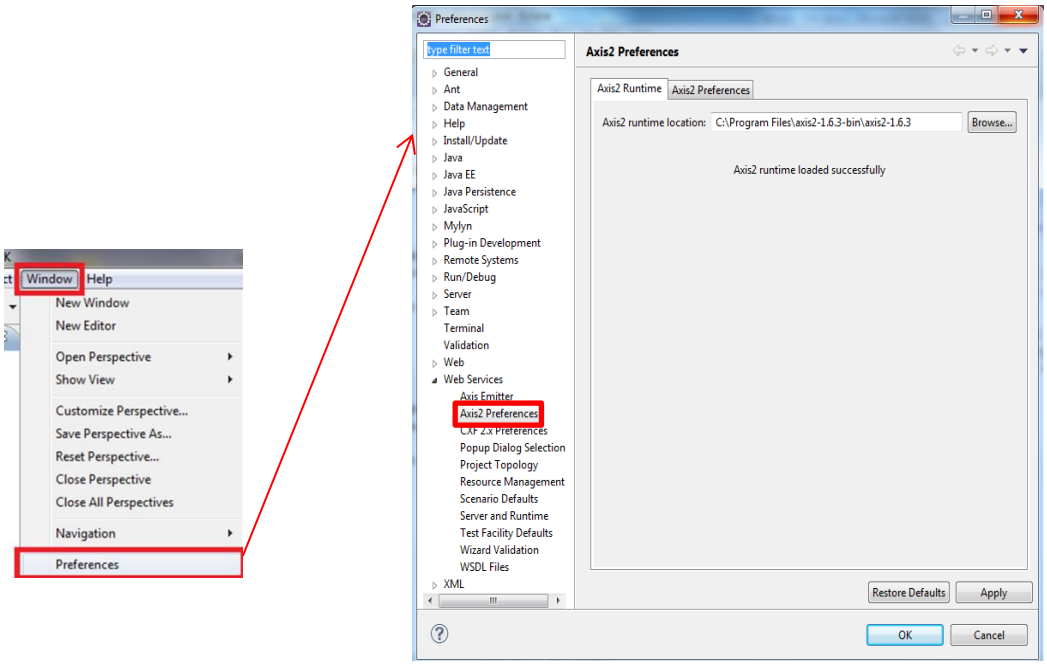


FIGURA 20: asignar Apache Axis2 como motor de servicios web.