UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

Escuela Técnica Superior de Ingenieros de Telecomunicación
Universidad Politécnica de Valencia

Geolocalización a través de feeds RSS

Proyecto Final de Carrera

Autor: Javier Gómez Romero

Director: Carlos Enrique Palau Salvador

A mis padres y hermana, por su apoyo incesante y por creer en mí.

A Carlos Palau, por darme la oportunidad de aprender y de realizar este proyecto.

A Laura, Guillermo y Álvaro, por ayudarme a levantarme y seguir adelante siempre.

Índice

Lista de Acrónimos

E.T.S.I.T.	Escuela Técnica Superior de Ingeniería de Telecomunicación
P.F.C.	Proyecto Final de Carrera
N.E.R.	Reconocimiento de Nombres de Entidades, del inglés Named Entity Recognition.
N.L.P.	Procesado Natural del Lenguaje, del inglés Natural Language Processing
R.S.S.	Really Simple Syndication

Capítulo 1. Introducción

1.1. Preámbulo

Internet ha cambiado para siempre la forma en la que tratamos la información, tanto el modo de generar contenidos como la manera de consumirlos.

Las facilidades que prestan Internet y el mundo conectado han derivado en disponer actualmente de la capacidad de generar información a cada instante, inmediatamente después del acaecimiento de un suceso, casi como un reflejo involuntario e incontrolable más propio del cuerpo humano. Una circunstancia que, unida al acceso libre, creciente y generalizado a la red, permite que los contenidos digitales mantengan un ritmo de generación con índices de incremento exponencial y cuya cota máxima parece ser todavía ilimitada.

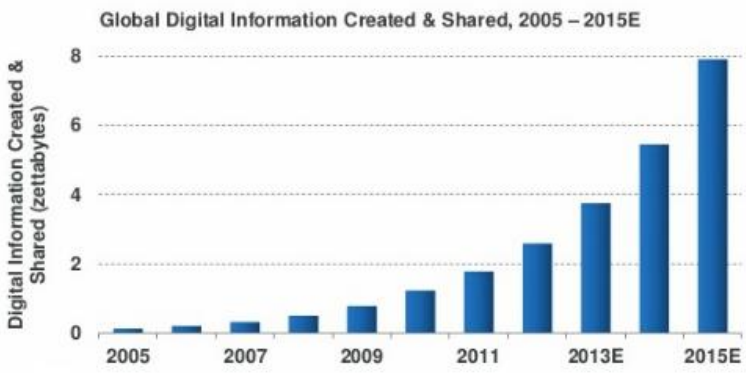

Figura 1.1 Evolución de generación de contenidos en Internet

Por ejemplo, teniendo en cuenta tan solo el famoso portal de videos youtube.com, se generan actualmente 100 horas de video por minuto y se consumen 4.000 millones de videos a diario en todo el mundo.

Del mismo modo, la era digital ha dado un vuelco a la forma en la que los usuarios consumen contenidos. Debido a la gran oferta de contenidos disponible, ahora son los usuarios los que eligen los contenidos en función de sus gustos y ellos son los que marcan su propio ritmo.

Ante esta nueva tesitura, ¿Cómo manejamos la ingente cantidad de información? "Organizar la información del mundo no es algo que puedan hacer ni 1000 personas." (Larry Page, cofundador y CEO de Google). Como bien plantea una de las voces más influyentes de la industria tecnológica, la organización es una labor que debe ser automatizada y puesta en manos de máquinas.

Una de las fórmulas de automatización que utilizan medios de comunicación y otros generadores de noticias es el uso del formato RSS con el objetivo de aglutinar de forma automática todo su flujo de comunicación bajo una única firma.

No obstante, ¿Qué ocurre si lo que pretendemos es extraer información clave de un flujo de noticias? Del mismo modo que humanamente es una tarea ardua e inmanejable por la abundante cantidad de información, se debe automatizar el proceso de extracción de información. Con el fin de analizar texto de forma automática se comenzaron a desarrollar técnicas de Procesado Natural del Lenguaje (NLP por sus siglas en inglés). El NLP es inteligencia artificial aplicada a la lingüística y cuyo objetivo es diseñar mecanismos capaces de comunicar e interpretar el lenguaje desde los ojos de un humano. El Procesado Natural del Lenguaje abre de esta forma las puertas de la extracción automática de información relevante contenida en textos mediante técnicas de Reconocimiento de Nombres de Entidades (NER). Las herramientas de reconocimiento de información actualmente permiten localizar y clasificar información relevante en categorías predefinidas.

Sin embargo, la extracción de información per se es insustancial si ésta no es susceptible de ser sintetizada, evaluada y no se es capaz de sacar conclusiones a partir de ella. Es por ello que, como apunta el escritor Umberto Eco: "toda información es importante si está conectada a otra." Por lo que georreferenciar la información nos aporta una visión espacial de dónde ocurren los sucesos, ofreciendo así una visión analítica automatizada de los puntos donde ocurren sucesos de interés.

1.2. Estado del arte

En este apartado se pasa revista a la situación actual de las tecnologías y objetivos que se desarrollan en el presente proyecto, cuya meta es la geolocalización de nombres de entidades reconocidos en flujos RSS.

1.2.1. Geolocalización

En el mundo de la información e Internet, la geolocalización auspicia una de las áreas tecnológicas de mayor expansión dentro del desarrollo software y a pesar de su gran popularidad sigue adquiriendo un protagonismo creciente.

Entendida la geolocalización como la identificación real geográfica de la posición de un objeto, ya sea éste un objeto virtual o un dispositivo real físico conectado a Internet. Se trata, por tanto, de dotar de una posición geográfica real a objetos virtuales susceptibles de poseer una representación física tangible.

Desde que Google en 2005 Google desarrollara Google Maps ofreciendo mapas dinámicos y navegables han aparecido tanto nuevas funcionalidades en el sector de la geolocalización como nuevos posibilidades. Además, no sólo la posibilidad de disponer de sistemas de geolocalización open-source ha contribuido en la vertiginosa expansión del sector, sino que unido al boom de los smartphones, la fácil accesibilidad a redes inalámbricas y el creciente mercado de aplicaciones para dispositivos móviles han abierto nuevos horizontes.

Una de las principales ventajas de la geolocalización y que además guarda una estrecha relación con los fines del proyecto actual es la capacidad de valorar y analizar geográficamente los focos de atención presentes en la red. Es decir, nos proporciona la capacidad de situar empresas, monumentos, eventos o elementos de cualquier índole.

En referencia a la geolocalización de elementos presentes en Internet se han desarrollado soluciones como GeoRSS.

GeoRSS es un conjunto de estándares para representar información geográfica mediante el uso de capas y está construido dentro de la familia de estándares RSS. En GeoRSS se puede proveer información geográfica en forma de coordenadas, líneas o polígonos. Los *feeds* que implementan GeoRSS están implementados de tal forma que son capaces de leer coordenadas geográficas y generar mapas con la representación de éstas.

Sin embargo, a pesar de la conveniente utilidad geográfica aportada, su uso no es extendido por el momento. Además, la información geográfica viene determinada por la entidad publicante, pudiendo resultar insuficiente e incluso en ocasiones restrictiva. [1]

1.2.2. NER

El Reconocimiento automático de Nombres de Entidades (NER) es un campo relativamente nuevo dentro del procesado natural de lenguaje (NLP), con un panorama muy extenso todavía por dilucidar y cuyos límites y capacidades potenciales a día de hoy se estiman incalculables.

La capacidad de extraer información relevante de forma estructurada de la red proporciona una nueva vía de desarrollo que permite establecer una relación directa y fiable entre la información extraída y una localización geográfica.

A día de hoy existen herramientas cuyo fin es la de perseguir este vínculo.

Geo-Coding es el nombre de un proyecto final de Carrera realizado en Italia en el cual se persigue representar geográficamente mediante servicios de mapas Web localizaciones detectadas mediante NER. No obstante, este proyecto no revela información sobre el origen de las entidades posteriormente reconocidas. Tan sólo especifica que son menciones reconocidas en texto plano. Además, este trabajo se centra principalmente en el modelo heurístico de las herramientas NER utilizadas y en distintas vías de representación geográfica de las entidades reconocidas.

En otro trabajo en el ámbito docente realizado en la Universidad Rey Juan Carlos de Madrid, se persigue localizar menciones de municipios en noticias dentro de *feeds* RSS mediante herramientas de Reconocimiento de Nombres de Entidades y posteriormente representarlos gráficamente.

También, otra solución de vinculación geográfica con Reconocimiento de Nombres de Entidades es la basada en la extracción de entidades de *feeds* de la red social Twitter y georreferenciarlos. [2]

1.3. Propósito final

La meta que persigue este proyecto es llevar un poco más lejos la relación entre localizaciones geográficas y la detección de menciones de entidades en Internet.

Las soluciones expuestas anteriormente son, por norma general, abstractas. Como norma general la utilización de las herramientas de Reconocimiento de Nombres de Entidades no se ejecutan sobre entidades conocidas. Es decir, aprovechan los modelos y clasificadores NER existentes sin crear uno propio y lo utilizan para clasificar entidades aleatorias y no basan sus objetivos en la identificación de identidades concretas. Igualmente, un rasgo común entre el panorama actual es el de únicamente representar geográficamente aquellas entidades que son propiamente ya localizaciones geográficas, sean éstas ciudades, coordenadas o cualquier elemento explícitamente de carácter geográfico.

Este proyecto, en cambio, desea acometer nuevas soluciones. Partiendo de la utilización de *feeds* RSS como fuente de información y mediante la creación de una serie de distintos modelos propios de clasificación a partir de herramientas de Reconocimiento de Nombres de Entidades (NER), se persigue vincular geográficamente las entidades reconocidas con posiciones geográficas. Además, los propósitos del proyecto no finalizan aquí. Se pretenden utilizar tres métodos distintos de Reconocimiento de Entidades con el fin de descubrir cuál es el método óptimo para identificar menciones concretas. Es decir, a diferencia de las soluciones actualmente existentes, se deberá abordar el desafío de localizar menciones específicas, realizar un estudio comparativo y determinar qué herramienta NER es la idónea y georreferenciar las menciones.

Como aliciente y debido a la cercanía de las Fallas de Valencia 2015 en los orígenes del presente proyecto, se toma la decisión de enfocar el proyecto a la localización de menciones de fallas de la ciudad de Valencia en flujos RSS. Tecnologías, servicios y herramientas utilizados.

Capítulo 2. Tecnologías, Servicios y Herramientas utilizados

Con la finalidad de acometer los objetivos propuestos y llevar a cabo la realización del proyecto se ha hecho uso de distintas tecnologías, herramientas y servicios.

2.1. Tecnologías

A continuación, se detalla la utilización de las diferentes tecnologías empleadas.

2.1.1. Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "*write once, run anywhere*"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para funcionar correctamente en otra. Esto es debido a que las aplicaciones de Java son compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (compañía posteriormente adquirida por el gigante tecnológico norteamericano Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos.

Entre sus características principales que lo han convertido en un lenguaje de éxito se encuentran las siguientes:

- **Seguro:** Tanto el lenguaje como la plataforma fueron diseñados en una estructura "*Bottom-Up*" siempre teniendo en cuenta la seguridad. Dada la naturaleza distribuida de Java, donde las applets se bajan desde cualquier punto de la Red, la seguridad se impuso como una necesidad de vital importancia. A nadie le gustaría ejecutar en su ordenador programas con acceso total a su sistema, procedentes de fuentes desconocidas. Así que se implementaron barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real.

- **Robusto:** Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros), ya que se ha prescindido por completo de los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria.

- **Distribuido:** Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

- **Portable:** La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas. Estas dos últimas características se conocen como la Máquina Virtual Java (JVM).

- **Multihilo:** A día de hoy, se ven considerablemente obsoletas aquellas aplicaciones que sólo pueden ejecutar una acción a la vez. Java soporta sincronización de múltiples hilos de ejecución (multithreading) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas. Así, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.

- **Orientado a objetos**: Todos los conceptos en los que se apoya esta técnica, encapsulación, herencia, polimorfismo, etc., están presentes en Java.

- **Dinámico:** El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.

No obstante, la plataforma Java separa sus recursos. Es decir, el entorno de ejecución de Java se distribuye de forma independiente respecto del entorno de programación.

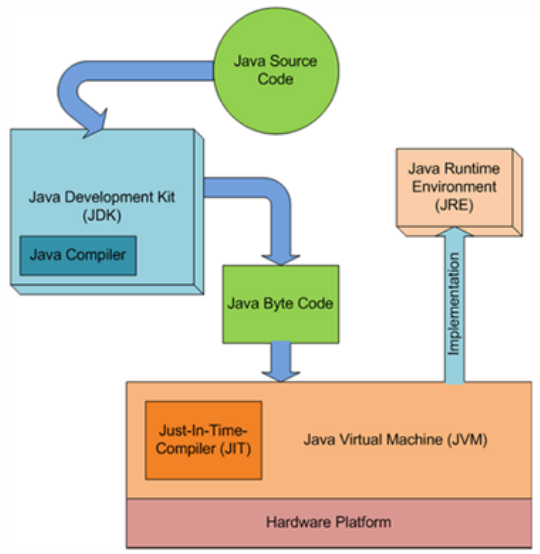

Figura 2.1 Interacción JDK - JRE - JVM

El JRE (*Java Runtime Environment*, o Entorno en Tiempo de Ejecución de Java) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes software o plugins (o conectores) en un navegador Web. Sun ofrece también el SDK de Java 2, o JDK (*Java Development Kit*) en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador.

Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK. [3][4]

2.1.2. JSON

JSON (*JavaScript Object Notation*) es un formato ligero para el intercambios de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. JSON nació como una alternativa a XML, el fácil uso en javascript ha generado un gran número de seguidores de esta alternativa. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

JSON está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un *objeto*, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

En JSON, se presentan de estas formas:

Un *objeto* es un conjunto desordenado de pares nombre/valor. Un objeto comienza con { (llave de apertura) y termine con } (llave de cierre). Cada nombre es seguido por : (dos puntos) y los pares nombre/valor están separados por , (coma). [5][6]

```
{"menu": {
   "id": "file",
   "value": "File",
   "popup": {
     "menuitem": [
       {"value": "New", "onclick": "CreateNewDoc()"},
       {"value": "Open", "onclick": "OpenDoc()"},
       {"value": "Close", "onclick": "CloseDoc()"}
     ]
   }
 }
```

Figura 2.2 Ejemplo de uso de formato JSON

2.1.3. XML

XML son las siglas del Lenguaje de Etiquetado Extensible. La expresión se forma a partir del acrónimo de la expresión inglesa *eXtensible Markup Language*. Se trata también de un lenguaje estándar recomendado por el *World Wide Web Consortium (W3C)*. Con la palabra "Extensible" se alude a la no limitación en el número de etiquetas, ya que permite crear aquellas que sean necesarias.

XML surgió como un lenguaje de marcado para sustituir a HTML. Ambos lenguajes son herederos de SGML, el lenguaje de marcas estándar para la descripción formal y de contenido de los documentos, no solamente para la presentación de dichos documentos. El desarrollo de XML comenzó en 1996 y desde entonces ha tenido un desarrollo exponencial. En realidad, XML surge del campo empresarial, ya que HTML era un lenguaje poco potente para soportar de un modo eficaz y masivo hacer negocios virtuales. Intentando mejorar HTML y tomando como punto de partida el viejo lenguaje SGML (*Standard Generalized Markup Language*), pero simplificándolo para poder trabajar en la Web, se creó XML y sólo 2 años después, en febrero de 1998, fue adoptado como recomendación por el World Wide Consortium, quien lanzó la versión 1.0. HTML tiene su propia especificación basada en XML, la del lenguaje XHTML (*eXtensible Hypertext Markup Language*) que es, en realidad, un paso intermedio de la migración de HTML hacia XML.

La primera definición de XML fue la de "Sistema para definir, validar y compartir formatos de documentos en la Web". Para crear XML se tomaron las mejores partes tanto del lenguaje SGML como del HTML. La diferencia fundamental entre HTML y XML es que el primero estaba orientado a la presentación de datos, mientras que XML está orientado a los datos en sí mismos, por lo que cualquier software informático trabajará mejor con XML. Sin duda, esta diferencia es fundamental para los nuevos desarrollos de la Web donde se da suma importancia al contenido de los datos y su tratamiento, y no sólo a su presentación.

HTML era, principalmente, un lenguaje de presentación que definía un conjunto de etiquetas y atributos válidos y que ofrecía un significado visual para cada elemento del lenguaje, por el contrario, XML no define las etiquetas ni cómo se utilizan, sino que ofrece un escaso número de reglas sintácticas para poder crear documentos. Así pues, XML no es un lenguaje, sino un metalenguaje o lenguaje para definir otros lenguajes. XML no sustituye a HTML puesto que sirven para cosas distintas: una cosa es presentar la información (para lo que sigue siendo válido HTML) y otra bien distinta es representar e intercambiar los datos de forma independiente a su presentación (que es para lo que sirve XML). Además, XML no sólo se aplica en Internet, sino que se propone como un lenguaje de bajo nivel para intercambio de información estructurada entre distintas plataformas. Se puede utilizar en bases de datos, hojas de cálculo, editores de texto, etc. y no sólo en la Web. HTML y XML son, pues, dos lenguajes complementarios.

Las diferencias fundamentales de XML con respecto a HTML son las siguientes: no requiere DTD (Document Type Definition), el XML tiene punteros a la estructura de los datos, lo que ahorra tiempo y simplifica el software de aplicación. XML no dispone de soporte para excepciones, por lo que cada etiqueta realiza siempre la misma función. Posee independencia de los navegadores y del sistema de objetos, porque en lugar de añadir etiquetas de presentación al documento se remite a una hoja de estilo realizada en XSL (Extensible Style Language).

XML es un lenguaje que permite jerarquizar y estructurar la información y describir los contenidos dentro del propio documento, así como la reutilización de partes del mismo. La información estructurada presenta varios contenidos (texto, imágenes, audio, etc.) y formas: hojas de cálculo, tablas de datos, libretas de direcciones, parámetros de configuración, dibujos técnicos, etc. La forma da alguna indicación de qué papel puede jugar el contenido (por ejemplo, el contenido de una

sección encabezada con un significado difiere del contenido de una nota a pie de página, lo que significa algo diferente que el contenido de un pie de foto o el contenido de una tabla de datos). Más o menos todos los documentos tienen la misma estructura.

Los programas que producen "datos estructurados" a menudo también permiten que estos datos puedan guardarse tanto en formato binario como en formato texto. El formato texto permite ver los datos sin el programa que los ha producido. El lenguaje XML se basa en el lenguaje Unicode (con un conjunto de caracteres de 16 bits, más que el formato ASCII). XML consiste de una serie de reglas, pautas o convenciones para planificar formatos de texto para tales datos, de manera que produzcan archivos que sean fácilmente generados y leídos por un ordenador, que sean inequívocos y que eviten los problemas más comunes como la falta de extensibilidad, la falta de interoperabilidad entre plataformas o la falta de soporte para universalizar su tratamiento. Los archivos XML son archivos de texto, pero más difíciles de leer por las personas que los archivos HTML. Se puede usar un editor de texto para programar XML, pero cualquier error u olvido de una etiqueta dejará inservible dicho archivo. El lenguaje XML es más estricto que el HTML. [7][8]

2.1.4. RSS

RSS son las siglas en inglés de *Really Simple Syndication* - cuyo significado traducido al español es "*publicación simultánea de contenidos en diferentes medios de forma muy simple*"-. RSS utiliza una familia de formatos "*feed*" estandarizados para publicar información actualizada, ya sea mediante entradas de blog, titulares, artículos completos, audio o video. Un documento rss – más conocido como "*feed*" o canal – incluye texto, ya sea completo o resumido, y metadatos, como por ejemplo fecha, autor o coordenadas geográficas (GeoRSS) entre otros.

Los *feeds* RSS permiten a la entidad publicante sindicar – publicar simultáneamente contenidos en diferentes medios – datos automáticamente. Un *feed* básicamente es un formato XML estándar que asegura la compatibilidad entre un gran rango de máquinas/programas.

Suscribirse a un *feed* permite al usuario no tener la necesidad de entrar en un sitio web de forma manual para comprobar si hay nuevos contenidos. En cambio, el navegador monitoriza constantemente el sitio web e informa al usuario de las actualizaciones. El

navegador puede recibir órdenes para descargar las nuevas actualizaciones de forma automática.

El software conocido como “lector RSS” o “agregador”, el cual puede ser una aplicación informática común de escritorio o una aplicación web o móvil incluso, muestra los datos RSS al usuario. El usuario se suscribe a un rss introduciendo la url del *feed* o haciendo click sobre el icono RSS correspondiente en el navegador. El lector RSS verifica los feeds agregados periódicamente para extraer los contenidos nuevos y descargarlos facilitando el proceso para el usuario. [9][10]

2.1.5. NLP

El procesamiento de lenguajes naturales — abreviado PLN, o NLP del idioma inglés *Natural Language Processing* — es un campo de las ciencias de la computación, inteligencia artificial y lingüística que estudia las interacciones entre las computadoras y el lenguaje humano. El PLN se ocupa de la formulación e investigación de mecanismos eficaces computacionalmente para la comunicación entre personas y máquinas por medio de lenguajes naturales. El PLN no trata la comunicación por medio de lenguajes naturales de una forma abstracta, sino de diseñar mecanismos para comunicarse que sean eficaces computacionalmente —que se puedan realizar por medio de programas que ejecuten o simulen la comunicación—. Los modelos aplicados se enfocan no sólo a la comprensión del lenguaje de por sí, sino a aspectos generales cognitivos humanos y a la organización de la memoria. El lenguaje natural sirve sólo de medio para estudiar estos fenómenos. [8]

Mediante el uso de aprendizaje automático se pueden desarrollar herramientas potentes de PLN. Los algoritmos modernos de PLN están basados en aprendizaje automático, concretamente aprendizaje estadístico automático. El paradigma del aprendizaje automático es diferente al de sus predecesores. Las implementaciones de aprendizaje automático anteriores se basaban en códigos escritos a mano en el que se incluían conjuntos de una numerosa cantidad de reglas. Actualmente, en cambio, se utilizan algoritmos de aprendizaje general, a menudo basados en procesos estadísticos y de esta forma, aprender automáticamente las reglas a partir del análisis de un *corpus* compuesto por ejemplos reales de texto. Un *corpus* es un conjunto de documentos, e incluso en ocasiones de frases sueltas, que ha sido anotado a mano referenciando los

valores de las palabras y de esta forma que el algoritmo aprenda automáticamente en función de los valores del *corpus*.

Multitud de diferentes tipos de algoritmos de aprendizaje automático se han aplicado en labores tareas PLN. Estos algoritmos cargan como entrada un amplio conjunto de características generadas a partir de los datos de entrada. Actualmente, los algoritmos están cada vez más basados en modelos estadísticos. Estos algoritmos toman decisiones basadas en procesos estocásticos y establecimiento de pesos a los distintos valores. Estos modelos poseen la ventaja de poder expresar con relativa certeza varias soluciones correctas en lugar de una única, otorgando así resultados más fiables y completos para sistemas de gran volumen.

Los sistemas basados en algoritmos de aprendizaje automático tienen, por norma general, una gran cantidad de ventajas respecto a aquellos con reglas establecidas "a mano":

- Los procedimientos de aprendizaje usados durante el aprendizaje automático se centran en los casos más comunes, mientras que con las reglas hechas "a mano" el esfuerzo suele ser más difuso y no se focaliza tanto.

- Los procesos de aprendizaje automático utilizan algoritmos que hacen uso de técnicas de estadística inferencial con el fin de crear modelos robustos independientes de los datos sobre los que se han generado – por ejemplo, en *corpus* que contienen palabras o estructuras nunca antes vistas o palabras mal escritas o gramaticalmente incorrectas -. Generalmente, llegar al nivel de precisión de los modelos estadísticos con reglas escritas "a mano" es una tarea ardua, requiere mayor tiempo y no siempre genera un resultado igualmente satisfactorio.

- Los sistemas basados en aprendizaje automático siempre son susceptibles de ser mejorados simplemente aumentando el volumen del *corpus*. Sin embargo, en aquellos escritos "a mano" se debe aumentar la complejidad de las reglas. Además, existe un límite en términos de complejidad en sistemas escritos a mano, ya que alcanzan un punto en el que se hacen inmanejables. Todo lo contrario que con los modelos de aprendizaje automático.

El desarrollo del "Natural Language Processing" ha dado lugar a un gran número de aplicaciones entre las cuales cabe destacar las siguientes: [11]

- **Resumir automáticamente**
- **Traducción automática**
- **Análisis morfológico, sintáctico o gramatical**
- **Reconocimiento automático de entidades**
- **Etiquetado gramatical**

2.1.6. Chunking

Definición en inglés y cuya traducción aproximada es Análisis Sintáctico superficial. Se trata de una técnica perteneciente a NLP de análisis sintáctico que identifica los elementos constituyentes de una frase, ya sean formas verbales, grupos nominales u otra clase, sin especificar sus estructuras internas ni su funcionalidad dentro de la frase.

2.1.7. NER

NER son las siglas inglesas para "*Named Entity Recognition*", es decir, reconocimiento de nombres de entidades. Es una subtarea de extracción de información derivada de métodos de "Natural Language Processing" cuyo objetivo reside en localizar y clasificar elementos pertenecientes a un texto en categorías predefinidas, tales como personas, organizaciones, localizaciones, valores monetarios, etc.

2.2. Servicios

A continuación, se detallan los servicios empleados en el presente proyecto.

2.2.1. LingPipe

LingPipe es un conjunto de herramientas de *NLP* de última generación desarrollado en Java por *Alias-I inc.* y desempeña funciones de '*tokenización*', detección de frases, reconocimiento de nombres de entidades (*NER*), clasificación y etiquetado *POS* entre las más destacables.

Basada en una arquitectura diseñada con el objetivo de ser eficiente, escalable, reutilizable y de gran robustez, LingPipe provee una API Java con módulos de código y unidades de prueba. Los modelos mencionados pueden ser en varios idiomas o géneros. Además, ofrece bases y tutoriales para crear modelos de entrenamiento de forma sencilla para nuevas tareas, salida "*n-best*" basada en estimaciones estadísticas y la posibilidad de hacer modelos sensibles a caracteres en mayúscula o minúscula.

Centrando la atención dentro del campo del Reconocimiento de Nombres de Entidades (*NER*), LingPipe propone varias soluciones. Existe la opción de clasificación mediante la creación previa de un modelo de entrenamiento estadístico supervisado, mediante la creación de un diccionario o también basándose en una "expresión regular", más conocida en el argot informático como *regex*.

- **Rule-Based LingPipe NER**

  El reconocimiento de nombres de entidades se simplifica notablemente si se encuentra un patrón común entre las entidades objetivo. Por ejemplo, si se buscan direcciones de correo electrónico cuya estructura es *cadena@cadena.cadena*.

  La estructura de una dirección de correo electrónico es conocida por lo que podemos acotarla apoyándonos en una regla *regex*, como la expresada a continuación:

```
[A-Za-z0-9](([_\.\-]?[a-zA-Z0-9]+)*)@([A-Za-z0-9]+)(([\.\-]?[a-zA-Z0-
9]+)*)\.([A-Za-z]{2,})
```

  Una vez definido el patrón, se crea una clase específica extendiendo la clase RegExChunker de la API de LingPipe en la que como atributos se especifican el Regex, el tipo de entidad y el "score", puntuación en español. Este último parámetro propio de la API de LingPipe mide la verosimilitud de un *chunk*. Si un *chunk* cumple exactamente los requisitos buscados, su *score* será 0.0.

  Finalmente, haciendo uso de esta clase se puede buscar nombres de entidades que definidos bajo el patrón predefinido en textos de forma automática.

- **LingPipe NER basado en “diccionario aproximado”**

Este es el modelo escogido y desarrollado en el presente proyecto dentro de los servicios de LingPipe y cuyas razones se detallan en el punto más adelante, en el punto 0.

La clase dict.ApproxDictionaryChunker perteneciente a la API de LingPipe implementa un chunker basado en la búsqueda aproximada de las entradas de un diccionario de referencia, aunque también ofrece la posibilidad de búsqueda exacta. Es decir, no sólo identifica coincidencias exactas sino que mediante unos parámetros ponderados realiza búsquedas aproximadas.

El desarrollo de este método se basa en la creación de un diccionario de referencia de los nombres de entidades objetivo, por lo que no realiza una búsqueda abstracta, sino que se centra en las entidades pertenecientes al diccionario.

En la creación del chunker, se establecen un conjunto de propiedades junto al diccionario que constituyen el motor de este método NER. Estas propiedades son valores numéricos sobre los que el chunker ejecutará sus decisiones para determinar si las palabras halladas en un texto se corresponden aproximadamente con las especificadas en un diccionario.

Las propiedades de especificación obligatoria son las seis detalladas a continuación:

match: Peso otorgado a un chunk cuya coincidencia es exacta.

delete: Peso otorgado a la eliminación de un carácter para buscar la coincidencia.

insert: Peso acreditado a la inserción de un carácter.

substitute: Peso asignado en caso de sustituir un carácter en búsqueda de coincidencias.

transpose: Peso acreditado a la transposición entre caracteres contiguos.

maxDistance: Límite máximo establecido de aproximación.

Entre estas propiedades, por cada operación realizada se suma el peso asignado y un chunk será identificado como coincidencia si realiza un número de operaciones cuya suma de pesos no supere el valor de maxDistance definido.

A modo de ejemplo:

En una entrada de diccionario tenemos la entidad “Duque de Gaeta Puebla de Farnals” y en un texto se encuentra la referencia “Duc de Gaeta Pobla de Farnals”. Nuestro chunker realizará modificaciones de sustitución, inserción o sustracción en las palabras “Duc” y “Pobla” y verificará la coincidencia siempre que encuentre una combinación de operaciones cuyas suma de pesos no supere el valor de maxDistance. [12]

2.2.2. Apache Open NLP

La librería de Apache NLP es un conjunto de herramientas de aprendizaje automático enfocadas al procesado natural de lenguaje en textos. Este compendio realiza las funciones principales de NLP tales como tokenización, segmentación de frases, etiquetado POS, chunking, reconocimiento de nombres de entidades (NER) y resolución de correferencia sintáctica. Estas tareas son, generalmente, la piedra angular de servicios de procesado más complejos y para los cuales Open NLP provee la base necesaria.

El proyecto Open NLP, actualmente perteneciente a la fundación Apache, tiene como objetivo crear un conjunto sólido de herramientas para desempeñar las funciones mencionadas anteriormente. Un objetivo complementario es el de construir modelos para varios idiomas, así como proveer recursos de “texto anotado” o corpus para la creación de modelos funcionales derivados de estos. Además, Apache ofrece en su web su API para integrar las funcionalidades de Open NLP en Java junto con una amplia y detallada documentación.

Para el caso que concierne a este proyecto, el buscador de nombres, es decir, la funcionalidad NER de Open NLP puede detectar nombres de entidades o números en texto. Para ser capaz de desempeñar esta función, el buscador de nombres necesita un modelo. Este modelo depende directamente del idioma y del tipo de entidad para los cuales ha sido entrenado mediante la previa creación de un corpus. Los proyectos Open NLP ofrecen una serie de modelos de buscadores de nombres pre-entrenados con *cuerpos* también disponibles y al alcance del desarrollador.

En relación a la búsqueda de nombres en texto plano, Apache detalla una serie de instrucciones con los pasos y especificaciones a seguir en su documentación publicada. Además, en la ejecución de estos pasos también se han tenido en cuenta recomendaciones de la comunidad de desarrolladores Stack Overflow.

En primer lugar, se ha de crear un corpus. Este corpus es un texto generado a partir de texto real y en ningún caso inventado, ya que su precisión está basada en el análisis sintáctico, en la identificación de "palabras gatillo" (palabras que aparecen recurrentemente antes o después de las entidades objetivo) y otros elementos de relevancia sintáctica o gramatical involucrados. Se especifica que el fichero del corpus debe ser de extensión .train, tener una longitud recomendable de 15.000 líneas, estar codificado en utf-8, a una frase por línea y las entidades envueltas en el formato **<START:entidad> nombre <END>** sin aparición de caracteres especiales inmediatamente antes o después de los envoltorios y se deben guardar espacios en blanco entre el nombre y el envoltorio. Un ejemplo de uso real en el corpus es el detallado a continuación:

```
En cambio <START:falla> Justo Vilar Plaza Mercado Cabanyal <END> es una falla un poco estridente.
```

Para identificar menciones se hace uso de la API proporcionada por Open NLP y en y se integra en Java. Las funciones NER de Open NLP siguen la tónica general de la API y para su funcionalidad básica, se exige un texto de entrada y el modelo NER. En relación con la salida que devuelve la clase, la API ofrece una amplia gama de posibilidades y es altamente manipulable para obtener los datos que necesitemos. [13][14]

2.2.3. Stanford NLP

La Universidad de Stanford tiene su propio grupo de desarrollo NLP en el que participan investigadores, doctorandos, programadores y estudiantes en búsqueda de mejorar las herramientas que ellos mismos diseñan y ofrecen de forma open-source un amplio abanico de herramientas NLP.

En Named-Entity Recognition comenzaron en 2005 lanzaron Stanford NER, una herramienta en constante evolución. Actualmente, Stanford NER es una implementación

en Java que etiqueta secuencias de palabras en un texto cuyos nombres pertenecen a alguna categoría en concreto, ya sea personas, empresas o moléculas. Stanford NER ofrece un potente chunker con multitud de características configurables. Se ofrecen además sistemas NER muy depurados de reconocimiento de localizaciones, personas y organizaciones.

A Stanford NER también se le conoce como CRFClassifier. El software provee una implementación general de una cadena lineal de modelos secuenciales de Campo Aleatorio Condicional (Conditional Random Field en inglés). Es decir, ofrece la posibilidad de entrenar tus propios modelos. Es posible usar el código provisto para generar modelos secuenciales para cualquier propósito concreto. En el caso concreto de este proyecto, se ha creado uno específico para identificar menciones de nombres de monumentos falleros en Valencia.

Además, cabe resaltar que el software ofrecido es similar al modelo estándar loval+Viterbi con la particularidad de que se le han añadido nuevas características basadas en similaridad distribucional.

Los pasos a llevar a cabo para crear un clasificador NER de una entidad propia son los siguientes:

En primer lugar, se ha de crear un corpus. Este corpus es un texto generado a partir de texto real y en ningún caso inventado, ya que su precisión está basada en el análisis sintáctico, en la identificación de "palabras gatillo" (palabras que aparecen recurrentemente antes o después de las entidades objetivo) y otros elementos en los que el software CRF se apoya para crear un modelo a posteriori un modelo robusto. El corpus tiene que tener un formato específico. Debe estar dispuesto en un fichero .tsv con codificación utf-8, a una línea por token y separado por tabulación en la misma línea, la categoría del token, en caso de ser un token irrelevante se marcará con una 'O':

```
El        O
Presidente      O
de        O
la        O
falla     O
Sueca     FALLA
```

Figura 2.3 Formato del corpus en Stanford NLP

A continuación, se debe crear un fichero .prop en el que se determinan ciertas propiedades relativas a la creación del trainer. En este fichero .prop se especifica el nombre del modelo que se va a crear, la estructura del corpus y las características propias NER con las que se quiere entrenar el modelo, todas ellas pertenecientes a la clase NERFeatureFactory, consultable en la url detallada a continuación:

http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/ie/NERFeatureFactory.html

Una vez establecido el archivo .prop y generado un corpus de suficiente longitud, se crea el trainer por línea de comandos con la siguiente orden:

```
java -cp stanford-ner.jar edu.stanford.nlp.ie.crf.CRFClassifier -prop nombrearchivo.prop
```

Figura 2.4 Orden a ejecutar en consola para crear modelo Stanford NER

Una vez creado el modelo, se puede utilizar para clasificar entidades en texto de dos formas. Una es a través de una GUI simple proporcionada por Stanford NLP Group y cuya funcionalidad se limita a cargar el modelo deseado, pegar el texto susceptible de ser analizado sobre una ventana de texto y ejecutar. Tras ejecutar, aparecen resaltadas las entidades identificadas en categorías por colores y cuya leyenda se encuentra visible en todo momento en el GUI.

No obstante, la capacidad de manejabilidad de esta salida resaltada en colores es nula, por lo que la segunda opción para clasificar las entidades es desarrollar en Java una clase que permita operar con la salida. [15][16]

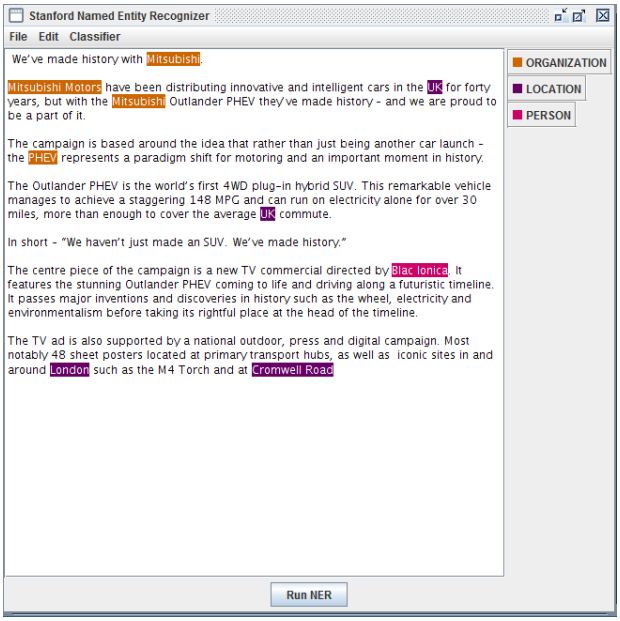

Figura 2.5 Captura de imagen de GUI de Stanford NLP

2.2.4. Google Static Maps

Google Maps es un servicio web de cartografía desarrollado por Google y lanzado en 2005 que ofrece perspectivas reales de calles, vista de tipo mapa o satelital de todo el planeta con todo tipo de detalle y cuya actualización se realiza a diario.

Esta potente aplicación cartográfica combina de manera inteligente sus principios técnicos y una gran accesibilidad para servir como una herramienta sumamente útil.

Varios aspectos de Google Maps son los responsables de su facilidad de uso por parte de cualquier programador: el sistema de deslizamiento de imagen, acoplado a la carga dinámica de nuevas imágenes, la interfaz minimalista. Además, entre las prestaciones más exitosas que ofrece al usuario se encuentra el planeador de rutas, poder cambiar de mapa en un clic o referenciar mapas.

Sin embargo, como el resto de aplicaciones de Google, GoogleMaps descansa poderosamente sobre la utilización de JavaScript, lo cual provoca que su uso en el presente proyecto quede restringido a poder mostrar localizaciones de forma estática. [17]

Para ello, se utiliza el API de Google Static Maps, el cual permite insertar imágenes de Google Maps sin utilizar JavaScript ni ningún sistema de carga dinámica de páginas. El servicio de Google Static Maps crea un mapa a partir de los parámetros de URL enviados a través de una solicitud HTTP estándar y genera una imagen de mapa en función de los parámetros proporcionados.

En el siguiente ejemplo se indica la URL de una imagen de mapa estático del centro de Nueva York, que se muestra debajo:

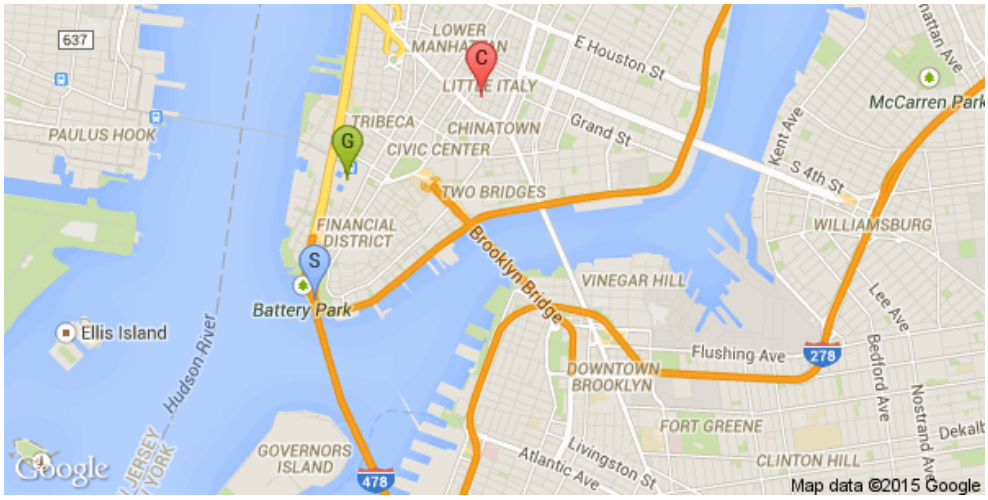

Figura 2.6 Captura de imagen de Nueva York con Google Static Maps

2.3. Herramientas

En este apartado se describen las herramientas utilizadas en el desarrollo del proyecto.

2.3.1. Eclipse

Eclipse es un entorno de desarrollo integrado (*IDE*), desarrollado originalmente por IBM como el sucesor de su familia de herramientas para K, Eclipse es a día de hoy desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y conjunto de productos complementarios, capacidades y servicios.

Se trata por tanto de una aplicación informática compuesta por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar "Aplicaciones de Cliente enriquecido". Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados, como el *IDE* de Java, llamado *Java Development Toolkit* (*JDT*) y el compilador (*ECJ*) que se entrega como parte de Eclipse. Además, Eclipse es una herramienta potente que no sólo sirve para desarrollar aplicaciones basadas en Java, sino que puede ser utilizada para desarrollar en otras muchas plataformas como *ABAP, C, C++, COBOL, Fortran, Haskell, JavaScript, Perl, PHP,* o *Python* entre otras.

La base para Eclipse es la Plataforma de cliente enriquecido (del Inglés *Rich Client Platform RCP).* Los siguientes componentes constituyen la plataforma de cliente enriquecido:

- Plataforma principal - inicio de Eclipse, ejecución de plugins
- El Workbench de Eclipse - vistas, editores, perspectivas, asistentes
- JFace - manejo de archivos, manejo de texto, editores de texto
- OSGi - una plataforma para bundling estándar.

Los widgets de Eclipse están implementados por una herramienta de widget para Java llamada *Standard Widget Toolkit,* a diferencia de la mayoría de las aplicaciones *Java*, que usan las opciones estándar *Abstract Window Toolkit (AWT*) o *Swing*. La interfaz de usuario de Eclipse también tiene una capa *GUI* intermedia llamada *JFace*, la cual simplifica la construcción de aplicaciones basadas en *SWT*.

El entorno de desarrollo integrado (*IDE*) de Eclipse emplea módulos (en inglés *plug-in*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos, donde las funcionalidades están todas incluidas las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente, Eclipse puede extenderse usando otros lenguajes de programación como son *C*, *C++* y *Python*, permitiendo también trabajar con lenguajes para procesado de texto como LaTeX o aplicaciones en red como Telnet y sistemas de gestión de base de datos. La arquitectura plug-in permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente con estos lenguajes, ya que soporta otros lenguajes de programación.

En cuanto a las aplicaciones clientes, Eclipse provee al programador con frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc.

Además, como característica adicional, Eclipse dispone de un editor de texto con un analizador sintáctico y compilación en tiempo real. Tiene pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (*wizards*) para creación de proyectos, clases, tests, etc., y refactorización. [18]

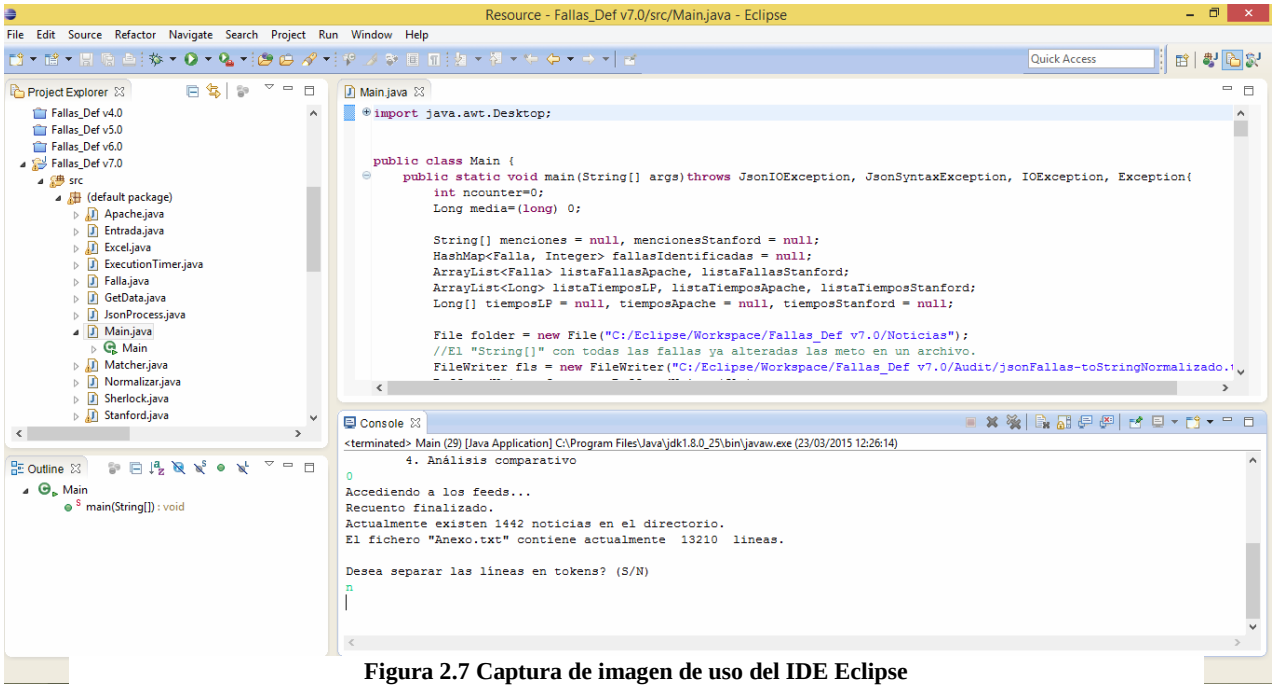

Figura 2.7 Captura de imagen de uso del IDE Eclipse

2.3.2. Microsoft Excel

Excel es una hoja de cálculo electrónica desarrollado por Microsoft, el cual se encuentra dentro del paquete de herramientas o programas ofimáticos llamados Office. El programa ofimático Excel es la hoja de cálculo electrónica más extendida y usada a nivel global, hoy en día el trabajo de cualquier ingeniero, financiero, matemático, físico o contable sería muy diferente sin la aplicación de cálculo Excel.

Una hoja de cálculo electrónica se define como un programa informático compuesto por columnas, filas y celdas, donde la intersección de las columnas y las filas son las celdas, en el interior de cada celda es el sitio donde podemos escribir cualquier tipo de información que posteriormente será tratada, siendo cada celda única en toda la hoja de cálculo. [19]

La principal ventaja del uso de las hojas de cálculo electrónicas reside en que es posible interconectar unas celdas con otras mediante el uso de funciones o reglas, de tal forma que si cambia el valor de una celda, automáticamente la hoja de cálculo electrónica recalculará y actualizará los valores de las celdas directamente relacionadas. Esta ventaja fue el origen y la base para impulsar el desarrollo de las hojas de cálculo electrónicas, debido a que antiguamente al no disponer de dicha herramienta informática, el trabajo de recalcular las diferentes hipótesis de un modelo matemático, físico o financiero representaba un tiempo y esfuerzo enorme, además del riesgo de caer en algún error durante el cálculo. Otras de las ventajas de Excel es la versatilidad y funcionalidad que presenta a la hora de realizar cualquier tipo de modelo, con esta potente herramienta informática podemos generar hojas de cálculo para el diseño y cálculo de estructuras civiles, gestión y control de la contabilidad de una empresa, gestión y control de los stocks de un almacén, diseños de modelos matemáticos, gestión de bases de datos, generación de presupuestos, planificación de proyectos, etc. Ofrece, por tanto, un amplio abanico de posibilidades. [10]

Capítulo 3. Diseño e implementación

En este bloque se va a proceder a describir detalladamente el proceso de creación del software y justificar el modelo diseñado para acometer los objetivos propuestos en el presente proyecto. Con este fin, a continuación se detalla el diseño y la forma de abordar su implementación partiendo de una visión más abstracta para terminar centrando la atención en detalles específicos.

Este capítulo ha sido seccionado en los siguientes subapartados: metodología, en el que se desarrolla la filosofía de diseño del software; un preludio, donde se explica de forma concisa el origen estructural de la arquitectura; una sección denominada arquitectura centrada en valorar el modelo completo con una perspectiva global; implementación, a su vez desplegada en varias secciones cuyo contenido atañe una visión detallada de los distintos módulos que forman el sistema.

3.1. Metodología

Este apartado pretende abordar la forma de trabajo o *framework* empleado para estructurar, planificar y controlar el proceso de desarrollo del sistema en el presente proyecto.

3.1.1. Elección de metodología

El framework para metodología de desarrollo de software consiste en:

- Una filosofía de desarrollo de programas de computación con el enfoque del proceso de desarrollo de software
- Herramientas, modelos y métodos para asistir al proceso de desarrollo software

Estos frameworks son, a menudo, vinculados con algún tipo de organización, que además desarrolla, apoya el uso y promueve la metodología.

Las metodologías de desarrollo de software tienen como principal objetivo presentar un conjunto de técnicas tradicionales y modernas de modelado de sistemas que permitan desarrollar software de calidad, incluyendo heurísticas de construcción y criterios de comparación de modelos de sistemas.

Para tal fin se describen, fundamentalmente, herramientas de Análisis y Diseño Orientado a Objetos (UML), sus diagramas, especificación, y criterios de aplicación de las mismas. Como complemento se describirán, en función del enfoque aplicado, las diferentes metodologías valoradas para el posterior desarrollo de software que utilizan dichas herramientas: [20] [21]

- ## Modelo en Cascada

Enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de todos los demás modelos de ciclo de vida.

La versión original fue propuesta por Winston W. Royce en 1970 y posteriormente revisada por Barry Boehm en 1980 e Ian Sommerville en 1985.

Ejemplo de una metodología de desarrollo en cascada es:

1. Análisis de requisitos
2. Diseño del Sistema.
3. Diseño del Programa.
4. Codificación
5. Pruebas
6. Verificación
7. Mantenimiento

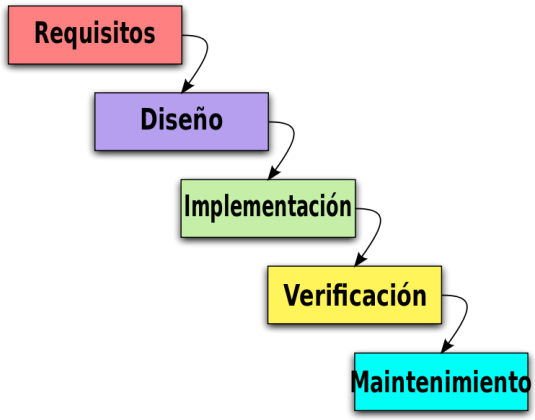

Figura 3.1 Metodología de desarrollo en cascada

De esta forma, cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo. La palabra *cascada* sugiere, mediante la metáfora de la fuerza de la gravedad, el esfuerzo necesario para introducir un cambio en las fases más avanzadas de un proyecto. [22]

- **Incremental**

Proceso de desarrollo de software creado en respuesta a las debilidades del modelo tradicional de cascada.

Básicamente este modelo de desarrollo, que no es más que un conjunto de tareas agrupadas en pequeñas etapas repetitivas (iteraciones), es uno de los más utilizados en los últimos tiempos ya que, como se relaciona con novedosas estrategias de desarrollo de software y una programación extrema, es empleado en metodologías diversas.

El modelo consta de diversas etapas de desarrollo en cada incremento, las cuales inician con el análisis y finalizan con la instauración y aprobación del sistema.

Se planifica un proyecto en distintos bloques temporales que se le denominan iteración. En una iteración se repite un determinado proceso de trabajo que brinda un resultado más completo para un producto final, de forma que quien lo utilice reciba beneficios de este proyecto de manera creciente.

Para llegar a lograr esto, cada requerimiento debe tener un completo desarrollo en una única iteración que debe de incluir pruebas y una documentación para que el equipo pueda cumplir con todos los objetivos que sean necesarios y esté listo para ser dado al cliente. Así se evita tener arriesgadas actividades en el proyecto finalizado.

Lo que se busca es que en cada iteración los componentes logren evolucionar el producto dependiendo de los completados de las iteraciones antecesoras, agregando más opciones de requisitos y logrando así un mejoramiento mucho más completo.

Una manera muy primordial para dirigir al proceso iterativo incremental es la de priorizar los objetivos y requerimientos en función del valor que ofrece el cliente.

La idea principal detrás de mejoramiento iterativo es desarrollar un sistema de programas de manera incremental, permitiéndole al desarrollador sacar ventaja de lo que se ha aprendido a lo largo del desarrollo anterior, incrementando, versiones

entregables del sistema. El aprendizaje viene de dos vertientes: el desarrollo del sistema, y su uso (mientras sea posible). Los pasos claves en el proceso son comenzar con una implementación simple de los requerimientos del sistema, e iterativamente mejorar la secuencia evolutiva de versiones hasta que el sistema completo esté implementado. En cada iteración, se realizan cambios en el diseño y se agregan nuevas funcionalidades y capacidades al sistema. [23]

Básicamente este modelo se basa en dos premisas:

- Los usuarios nunca saben bien que es lo que necesitan para satisfacer sus necesidades.
- En el desarrollo, los procesos tienden a cambiar.

El proceso en sí mismo consiste de:

- Etapa de inicialización
- Etapa de iteración
- Lista de control de proyecto

▪ Prototipado

Modelo de desarrollo evolutivo. El prototipo debe ser construido en poco tiempo, usando los programas adecuados y no se debe utilizar muchos recursos.

El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles para el cliente o el usuario final. Este diseño conduce a la construcción de un prototipo, el cual es evaluado por el cliente para una retroalimentación; gracias a ésta se refinan los requisitos del software que se desarrollará. La interacción ocurre cuando el prototipo se ajusta para satisfacer las necesidades del cliente. Esto permite que al mismo tiempo el desarrollador entienda mejor lo que se debe hacer y el cliente vea resultados a corto plazo. [24][25]

Las fases que comprende el método de desarrollo orientado a prototipos serían:

- Plan rápido
- Modelado, diseño rápido
- Construcción del prototipo
- Desarrollo, entrega y retroalimentación
- Comunicación
- Entrega del desarrollo final

- **Espiral**

El desarrollo en espiral es un modelo de ciclo de vida del software definido por primera vez por Barry Boehm en 1986. Las actividades de este modelo se conforman en una espiral, en la que cada bucle o iteración representa un conjunto de actividades. Las actividades no están fijadas a ninguna prioridad, sino que las siguientes se eligen en función del análisis de riesgo, comenzando por el bucle interior.

En cada vuelta o iteración hay que tener en cuenta:

- **Objetivos:** qué necesidad debe cubrir el producto.
- **Alternativas:** las diferentes formas de conseguir los objetivos de forma exitosa, desde diferentes puntos de vista como pueden ser:
- **Características:** experiencia del personal, requisitos a cumplir, etc.
- **Formas de gestión del sistema**.
- **Riesgo asumido con cada alternativa**.
- **Desarrollar y Verificar:** Programar y probar el software.

Si el resultado no es el adecuado o se necesita implementar mejoras o funcionalidades:

- Se planificaran los siguientes pasos y se comienza un nuevo ciclo de la espiral. La espiral tiene una forma de caracola y se dice que mantiene dos dimensiones, la radial y la angular:
    1. **Angular:** Indica el avance del proyecto del software dentro de un ciclo.
    2. **Radial:** Indica el aumento del coste del proyecto, ya que con cada nueva iteración se pasa más tiempo desarrollando.

Este sistema es muy utilizado en proyectos grandes y complejos como puede ser, por ejemplo, la creación de un Sistema Operativo.

Al ser un modelo de Ciclo de Vida orientado a la gestión de riesgo se dice que uno de los aspectos fundamentales de su éxito radica en que el equipo que lo aplique tenga la necesaria experiencia y habilidad para detectar y catalogar correctamente los riesgos. [26]

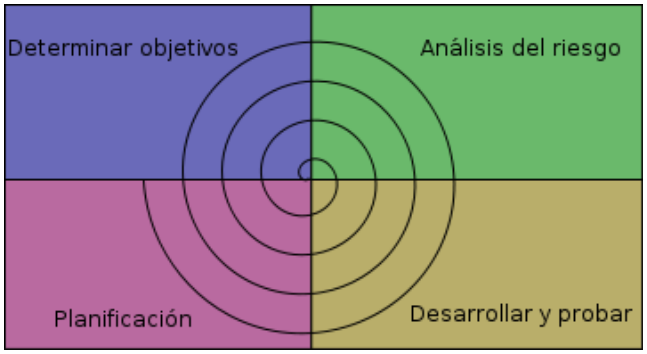

Figura 3.2 Ciclo de metodología espiral

Para cada ciclo habrá cuatro actividades:

1. **Determinar Objetivos.**
2. **Análisis del riesgo.**
3. **Desarrollar y probar.**
4. ***'Planificación.'***

▪ Programación Extrema (XP)

Metodología desarrollada por Kent Beck, autor del primer libro sobre la materia, *Extreme Programming Explained: Embrace Change* (1999). Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Este modelo defiende que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

La filosofía adoptada en programación extrema se basa en cuatro pilares fundamentales: simplicidad, comunicación, retroalimentación (feedback) y coraje.

▪ Simplicidad

El diseño debe simplificarse para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones por parte de diferentes desarrolladores hace que la complejidad aumente exponencialmente.

Para mantener la simplicidad es necesaria la refactorización del código, ésta es la manera de mantener el código simple a medida que crece.

- **Comunicación**

La comunicación se realiza de diferentes formas. Para los programadores el código comunica mejor cuanto más simple sea. Si el código es complejo hay que esforzarse para hacerlo inteligible. El código autodocumentado es más fiable que los comentarios ya que éstos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse sólo aquello que no va a variar, por ejemplo el objetivo de una clase o la funcionalidad de un método.

Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de cómo utilizar su funcionalidad. La comunicación con el cliente es fluida ya que el cliente forma parte del equipo de desarrollo. El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas.

- **Retroalimentación**

Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante.

Considérense los problemas que derivan de tener ciclos muy largos. Meses de trabajo pueden tirarse por la borda debido a cambios en los criterios del cliente o malentendidos por parte del equipo de desarrollo. El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Por ejemplo, las pruebas unitarias informan sobre el estado de salud del código. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

- **Coraje**

> Muchas de las prácticas implican valentía. Una de ellas es siempre diseñar y programar para hoy y no para mañana. Esto es un esfuerzo determinante a la hora de evitar estancarse en el diseño y requerir demasiado tiempo y trabajo para implementar el resto del proyecto. La valentía le permite a los desarrolladores que se sientan cómodos con reconstruir su código cuando sea necesario. Esto significa revisar el sistema existente y modificarlo si con ello los cambios futuros se implementaran más fácilmente. Además, valentía significa persistencia: un programador puede permanecer sin avanzar en un problema complejo por un día entero, y luego lo resolverá rápidamente al día siguiente, sólo si es persistente.

Tras un análisis a priori de las distintas filosofías y metodologías de desarrollo, se estableció la Programación Extrema como la corriente más afín a los principios del autor de este mismo proyecto.

Se estima que, por lo general, en el desarrollo de un software rara vez se sigue una secuencia lineal. Esto crea una mala implementación del modelo, lo cual hace que lo lleve al fracaso. Por ello, se decide descartar el modelo en cascada, en el que una etapa determinada del proyecto no se puede llevar a cabo a menos de que se haya culminado la etapa anterior. Igualmente, se desestima basar el desarrollo en un modelo de prototipos. En aras de desarrollar rápidamente el prototipo, el desarrollador suele tomar a menudo ciertas decisiones de implementación poco convenientes (por ejemplo, elegir un lenguaje de programación incorrecto porque proporcione un desarrollo más rápido). Decisiones que a medio o largo plazo pueden llegar a convertirse en un lastre.

Así las cosas, parece evidente que las metodologías clásicas no se ajustan a las exigencias actuales de desarrollo, si bien es cierto que el modelo en espiral es un modelo atractivo. No obstante, las metodologías ágiles pueden corregir ciertas contrariedades y mejorar las deficiencias del modelo en espiral mediante un desarrollo incremental del producto y un crecimiento adaptativo, según las necesidades que van surgiendo. Por tanto, se concluye que el modelo de desarrollo a seguir es el de Programación Extrema. [27]

3.2. Preludio

Este proyecto surge con relativa anterioridad a las fiestas de las Fallas de Valencia 2015 con la idea de desarrollar una herramienta software diseñada en Java, capaz de **posicionar geográficamente** nombres de **monumentos falleros** previamente reconocidos en noticias dentro de **flujos RSS** mediante tres métodos distintos de **NER** y, finalmente, realizar un **estudio comparativo** basado en el rendimiento de los tres métodos.

Igualmente, a raíz de la creación de los tres métodos de reconocimiento de entidades y su posterior estudio comparativo, se ha colaborado en pos de la mejora de la aplicación social para dispositivos móviles LiveFallas[1], desarrollada por la empresa Prodevelop en colaboración con el Grupo de Sistemas y Aplicaciones de Tiempo Real Distribuido de la Universitat Politècnica de València (SATRD-UPV)

En función de este entorno, se ha adecuado la estructura del software con el fin de cumplir con las especificaciones detalladas y mantener una arquitectura sencilla, adecuada y detallada en los apartados posteriores.

[1] https://www.upv.es/noticias-upv/noticia-7291-live-fallas-201-es.html

3.3. Arquitectura

En esta sección se detalla el diseño de la arquitectura de la aplicación creada.

3.3.1. Escenario global

En la imagen mostrada a continuación (Figura 3.3) se presenta un diagrama de bloques a modo de representación de la arquitectura del software.

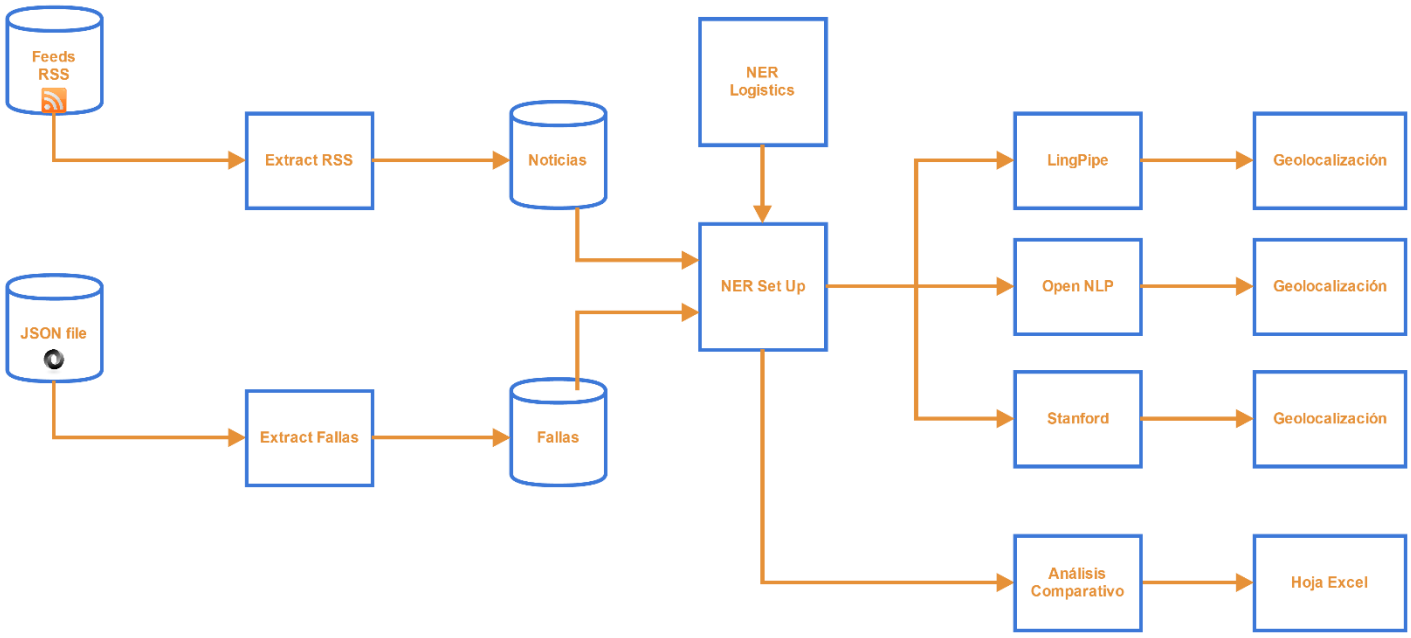

Figura 3.3 Arquitectura global del sistema

Se puede observar que se ha desarrollado un modelo secuencial en cascada.

El sistema parte de dos fuentes de información claramente diferenciadas. Por un lado, los distintos feeds RSS son los que suministran al sistema de información analizable y de la cual poder extraer información relevante. Es decir, es a partir de los flujos RSS de donde el sistema se nutre para recabar nueva información y ser posteriormente inspeccionada. Por otro lado, la web corporativa del Ayuntamiento de Valencia ofrece un catálogo de diferentes datos abiertos a disposición de desarrolladores en el enlace

detallado a pie de página.[2] Dentro del catálogo ofrecido, este software se hace acopio del fichero de formato JSON en el que se incluyen todos los datos relativos a los monumentos falleros de la ciudad de Valencia.

Una vez obtenidas y procesadas ambas fuentes de información, el sistema acondiciona los datos para, posteriormente, o bien ser examinados mediante uno de los tres métodos NER y a continuación geolocalizar los monumentos falleros reconocidos, o bien ser examinados por los tres métodos NER secuencialmente y obtener un estudio estadístico comparativo volcado en una hoja de cálculo Excel.

3.3.2. Cronología de desarrollo

Este apartado pretende exponer la evolución cronológica del sistema sustentada mediante una filosofía de programación extrema.

Con este fin, en el presente proyecto se han desarrollado los siguientes principios de programación extrema:

- El flujo de creación ha evolucionado siguiendo una **pauta iterativa y creciente**. A medida que se cumplían los objetivos iniciales, se continuaba modificando el sistema dotándolo de nuevas funcionalidades. De esta forma, el sistema ha sufrido una notable evolución desde que germinara con funciones primarias.

- A lo largo de todo el proceso se han ejecutado numerosas **pruebas unitarias**, poniendo a prueba el correcto funcionamiento de cada módulo implementado, verificando su correcto uso tanto a nivel aislado como ensamblado en el conjunto global de la arquitectura y, de esta forma, disponer de un sistema de **corrección de errores** con el fin de depurar el software final.

- En el caso actual concreto de tratarse de un trabajo académico, el tutor del PFC es el que adopta el rol de cliente y por ende, la figura que marca tanto objetivos como requisitos del software. Esta tesitura por tanto, sugiere que la integración entre el programador (alumno) con el cliente (tutor) es total, manteniendo un flujo de comunicación constante y reuniones periódicas entre las partes.

[2] http://goo.gl/BvDa6s

- Igualmente, debido a las continuas actualizaciones y mejoras en los distintos módulos del sistema en distintas fases del desarrollo, la **refactorización del código** es una constante. No obstante, al refactorizar código se ha de garantizar que no se produce una involución. Esta meta se consigue principalmente manteniendo el comportamiento de los módulos del sistema y no introduciendo fallos.

Sin embargo, a pesar de que el desarrollo se basa en programación extrema, debido a la situación excepcional de ser un Proyecto Final de Carrera y en consecuencia tratarse de un autor único, no se ha desarrollado ni programación en parejas ni se extiende la propiedad y desarrollo compartidos del código.

A continuación, en una serie de figuras se expone la evolución cronológica que ha padecido el sistema, donde mediante las características detalladas anteriormente han sido dispuestas al servicio del óptimo desarrollo de la herramienta:

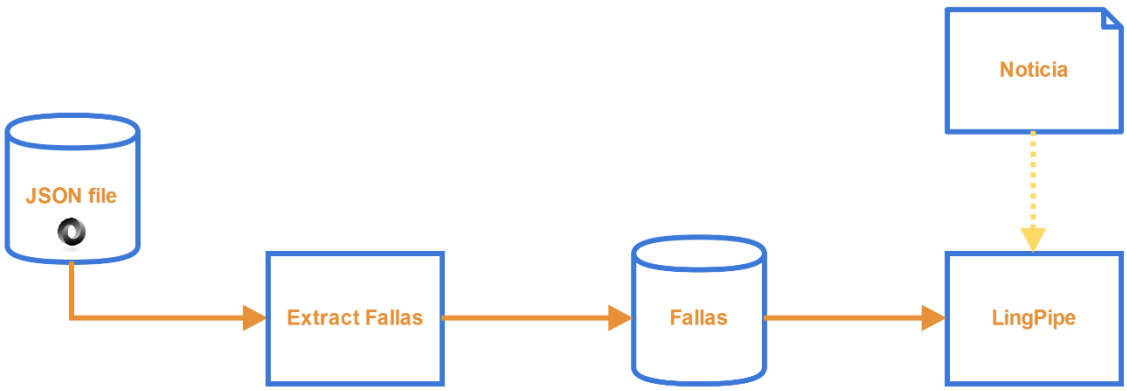

Figura 3.4 Herramienta Fallas GEO-NER v1.0

El germen del software partió como la extracción de los datos dentro del fichero JSON y de un desarrollo básico de la herramienta NER LingPipe, al cual se le pasaba una única noticia sobre la que se hacían distintas pruebas.

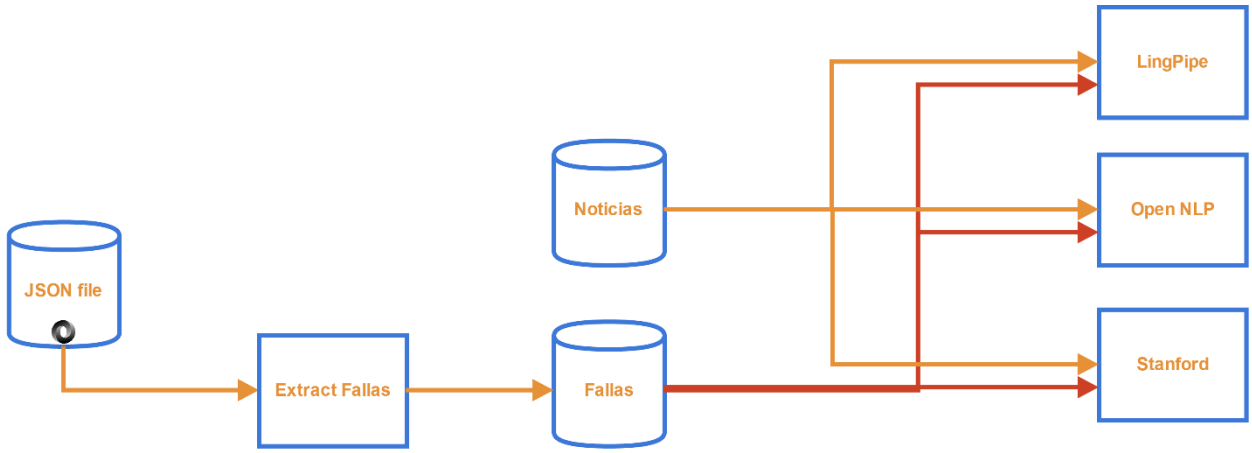

Figura 3.5 Herramienta Fallas GEO-NER v2.0

Posteriormente, se crearon los módulos de NER restantes, Open NLP y Stanford NLP. También, el número de noticias relacionadas con las Fallas crecía y se realizó una búsqueda de noticias de Fallas de otros años para hacer pruebas.

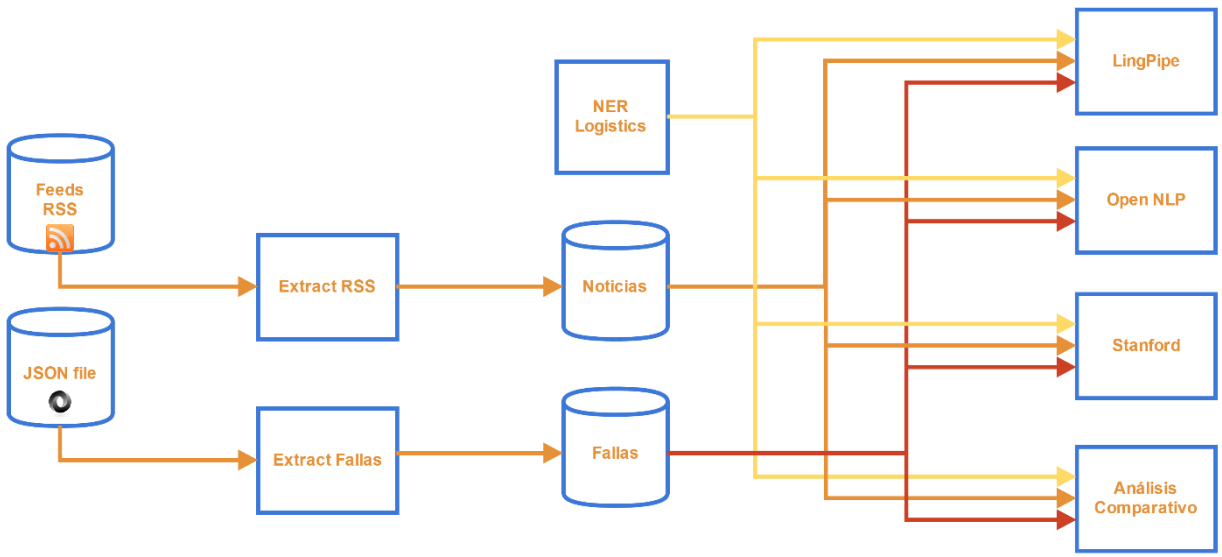

Figura 3.6 Herramienta Fallas GEO-NER v3.0

Tras la creación de los tres módulos NER, era preciso crear modelos propios de clasificación y reconocimiento específicos para las Fallas. Por esta razón, se decide incorporar un módulo lector de feeds RSS mediante el cual suministrar el necesario volumen de información para poder generar los mencionados modelos de forma satisfactoria. El módulo denominado "NER logistics" es el que realiza funciones de acondicionamiento de las noticias y entrenamiento de los modelos NER.

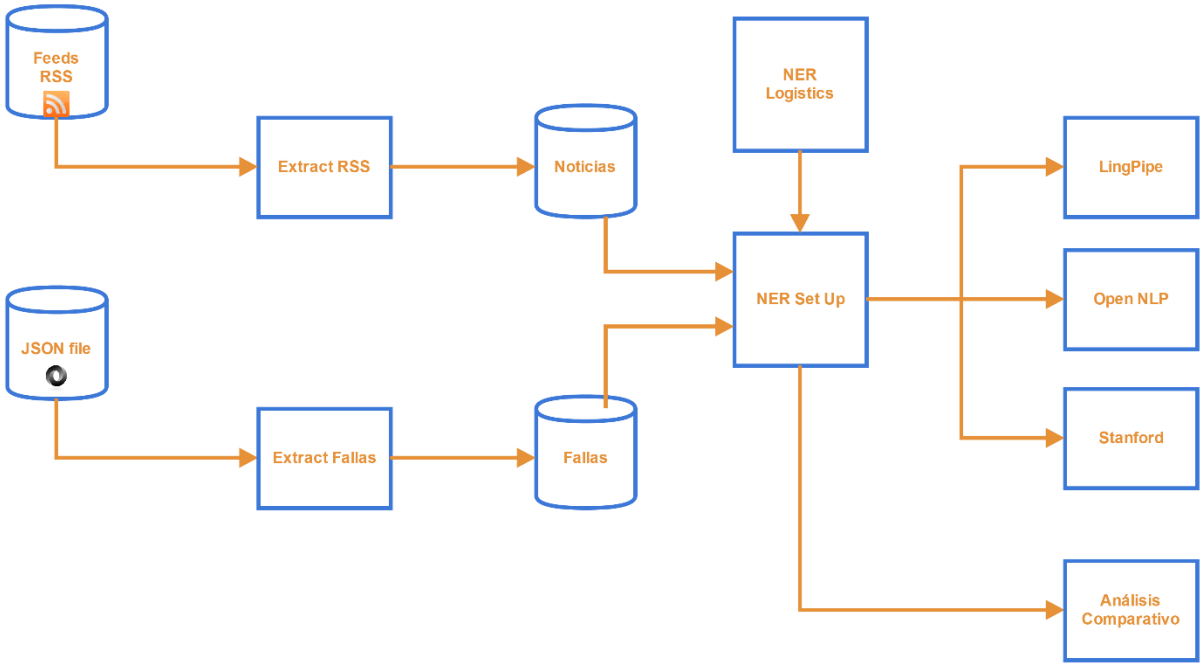

Figura 3.7 Herramienta Fallas GEO-NER v4.0

Finalmente, el motor de la herramienta ya toma el cuerpo definitivo donde el proceso de reconocimiento de nombres de entidades pasa en primer lugar por una fase de preacondicionamiento y, posteriormente, ya se ejecutan o bien uno de los tres métodos NER o bien los tres secuencialmente para su posterior estudio comparativo.

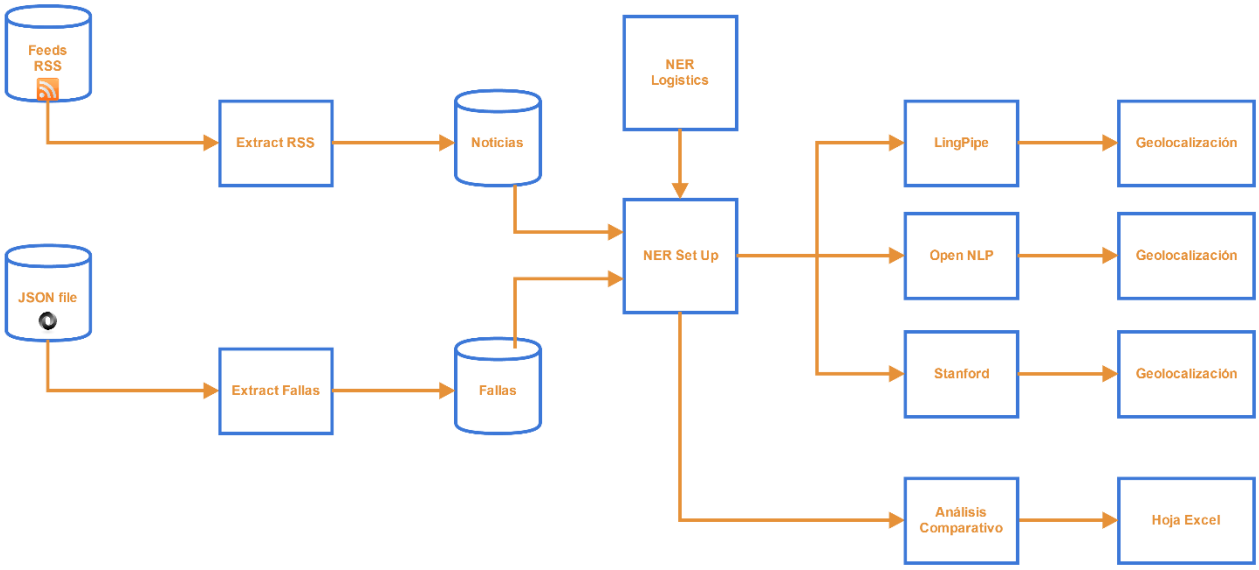

Figura 3.8 Herramienta Fallas GEO-NER en su versión final

Como último paso, se perfecciona y depura el sistema y se implementan funcionalidades de geolocalización y análisis estadístico mediante el volcado de información a una hoja de cálculo Excel.

3.3.3. Criterios de diseño

En toda actividad o desafío nuevo emprendido aparecen obstáculos o trabas que obligan a modificar ideas iniciales y tomar decisiones que repercuten directamente sobre el resultado final. Esta sección se centra en justificar determinados criterios de diseño y decisiones ejecutadas en función de las necesidades requeridas.

A continuación se expone una relación de aquellas decisiones que, por acción u omisión, han conferido en la estructura definitiva del sistema:

- **No creación de Base de Datos relacional y servidor**

  Al tratarse de un trabajo de carácter académico y por ende una herramienta no desarrollada dentro de un ámbito profesional, por simplicidad se ha descartado la adecuación de la arquitectura al uso de bases de datos relacionales para el manejo de datos privados y de uso recurrente, así como de dotar a la herramienta de un servidor con propósitos de actualización automática y ejecución autosuficiente.

- **Uso de lector de feeds RSS propio**

  La creación de un lector propio de feeds RSS nace a partir de la necesidad de manejar grandes volúmenes de información y de la condición indispensable de creación - o entrenamiento - de modelos NER propios de reconocimiento de nombres de monumentos falleros.

- **API Excel**

  Se hace uso de una API pública open-source para la creación de hojas de cálculo Microsoft Excel con el objetivo de automatizar el proceso de generación de una hoja de cálculo a partir de los datos obtenidos.

3.4. Implementación

En este apartado se describe el funcionamiento interno del software y su implementación. Con este propósito, se parte de una descripción desde una perspectiva a nivel global del software donde se define la implementación de la herramienta y su funcionamiento en conjunto para, a continuación, diseccionar la estructura completa en diversos módulos – destacados en forma de recuadro verde en la figura a continuación – y en cuyas secciones específicas se detalla el cometido y funcionamiento particular de cada uno. Por último, se detalla la relación de las distintas librerías empleadas en el desarrollo del proyecto.

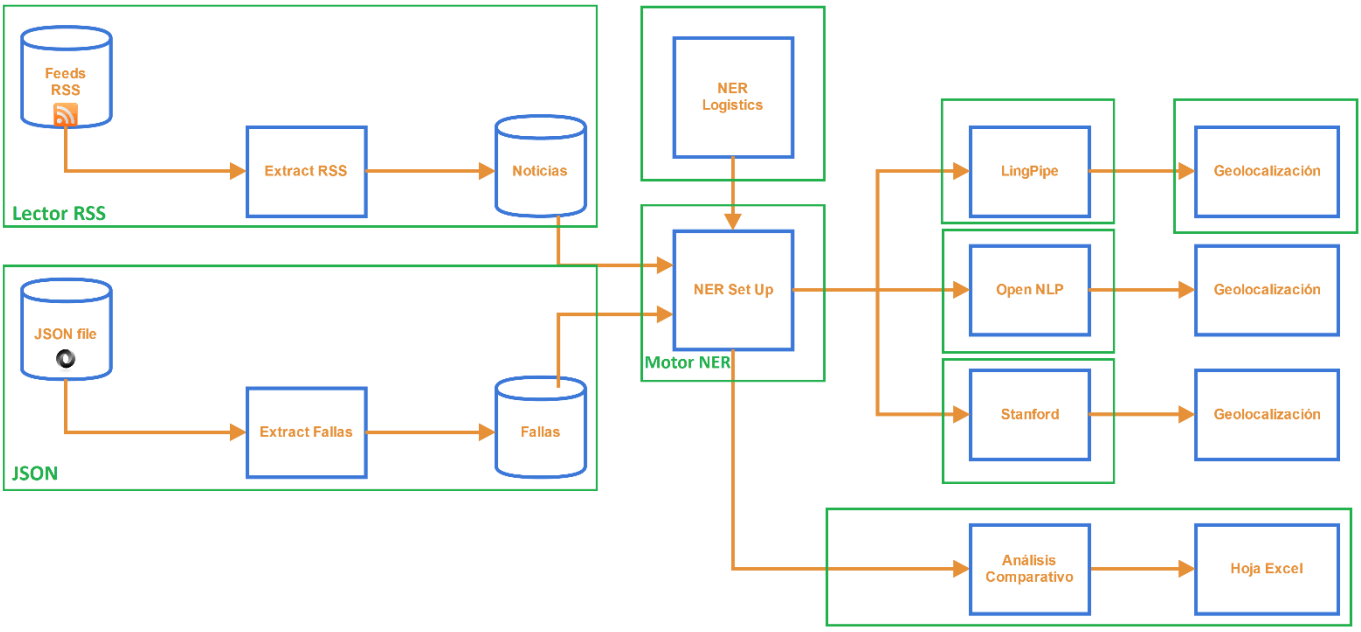

Figura 3.9 Arquitectura mostrada por módulos de implementación

Como puede apreciarse en la imagen que precede al texto, se ha desarbolado el sistema en nueve módulos distintos:

- **Lector RSS**
- **JSON**
- **NER Logistics**
- **Motor NER**
- **LingPipe**
- **Open NLP**
- **Stanford**
- **Análisis comparativo**
- **Geolocalización**

3.4.1. Diagrama de clases UML

En esta sección se detalla cómo se ha decidido implementar la herramienta software siguiendo el esquema de bloques de la arquitectura de la aplicación mediante pseudo-notación UML.

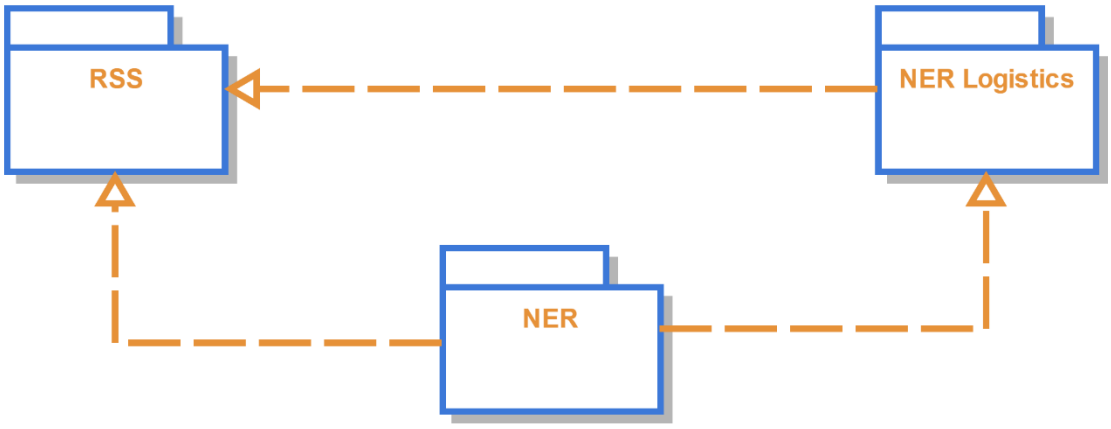

Figura 3.10 Interacción entre los distintos paquetes

En la imagen anterior (Figura 3.10) se muestra la relación entre los tres paquetes que forman el sistema. El paquete RSS contiene las clases pertenecientes al módulo "lector RSS"; NER Logistics, cuya misión es la de ejercer funciones de generación de corpus para posteriormente crear modelos de clasificación NER; finalmente, en el centro de la figura, se aprecia el paquete NER, verdadero motor del sistema y lugar donde se desarrolla la actividad principal de la herramienta.

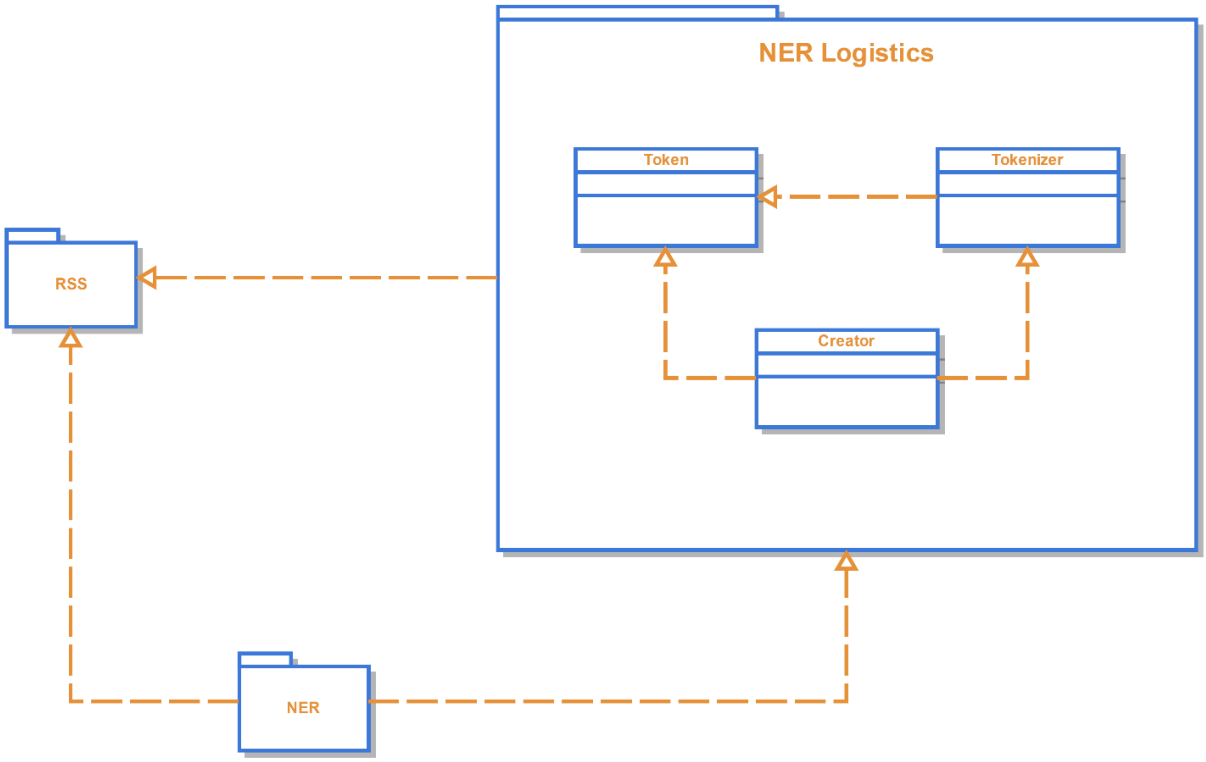

Figura 3.11 Relación de NER Logistics con los otros paquetes

En la imagen superior se presenta tanto la estructura interna del paquete NER Logistics así como la interacción que el propio paquete guarda con respecto a los otros dos paquetes. NER Logistics es un paquete cuyas funciones tienen un tratamiento especial en el proyecto y nace únicamente para cubrir necesidades surgidas en la elaboración del proyecto. NER Logistics recoge información de un fichero generado por el paquete RSS y posteriormente, a partir de un corpus preparado para Apache Open NLP es capaz de transformarlo en un corpus Stanford NLP y almacenarlo localmente. Por esta razón, se estima que la relación que guarda este paquete con el resto no es directa, sin embargo, el vínculo entre las clases es evidente.

En el módulo Lector RSS – el paquete RSS en Figura 3.10 – es donde se realizan todas las funciones relacionadas con el estándar RSS y la obtención de noticias relacionadas con las Fallas. Es el único proveedor de información de la herramienta y mantiene dos nexos de comunicación con el paquete principal NER, uno directo y otro indirecto. Por un lado, un vínculo indirecto ya que el punto de enlace entre el lector RSS y el paquete NER es mediante un directorio local donde el lector RSS vuelca las noticias

extraídas y desde el cual el paquete RSS carga las noticias. Por otro lado, al haber implementado el sistema entorno a un switch, una de cuyas opciones es la de leer nuevos feeds, se establece así una relación directa al hacer la llamada desde el paquete NER.

A continuación, se aportan los diagramas en pseudo-notación UML de la implementación de los paquetes RSS y NER.[3]

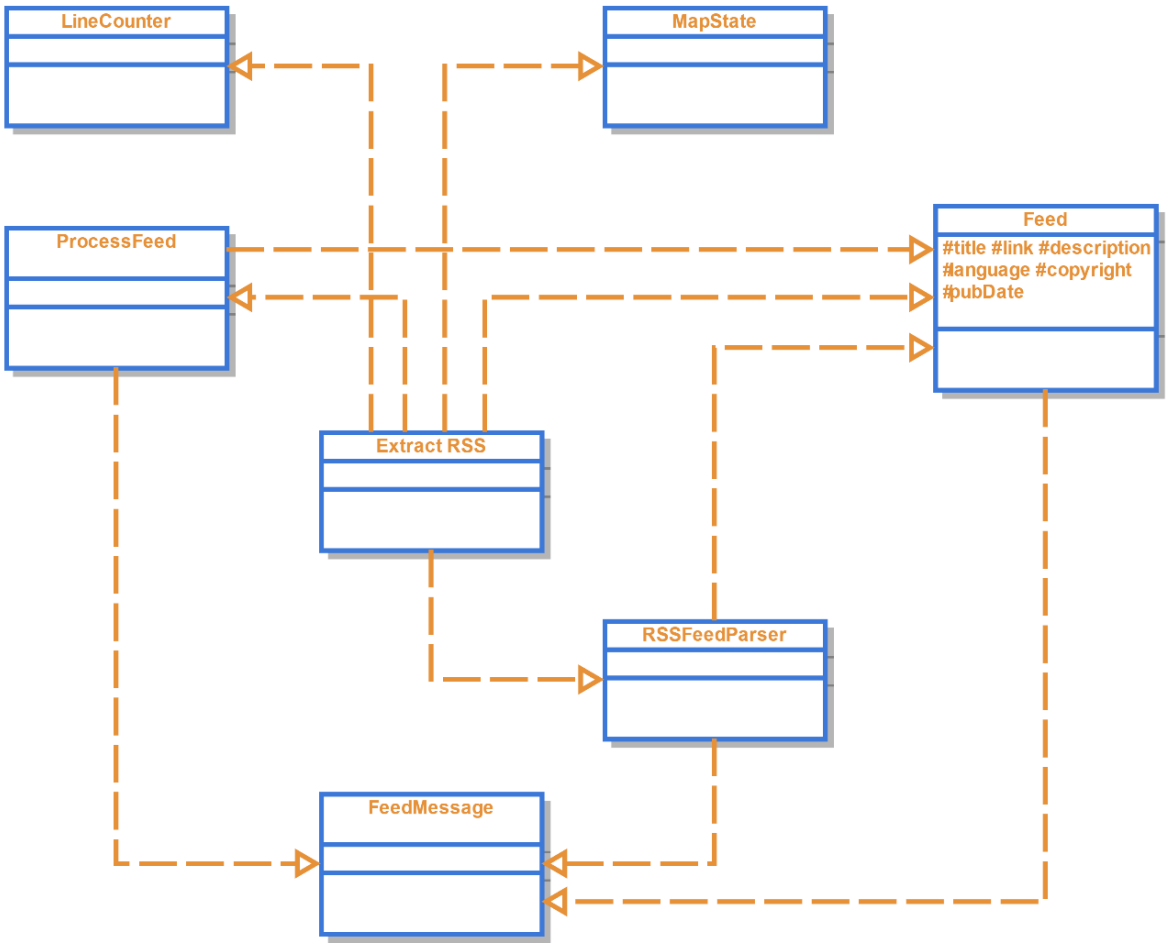

Figura 3.12 Relación de clases del paquete RSS

[3] La explicación detallada de la implementación y uso de las distintas clases se realiza en el apartado 3.4 Implementación.

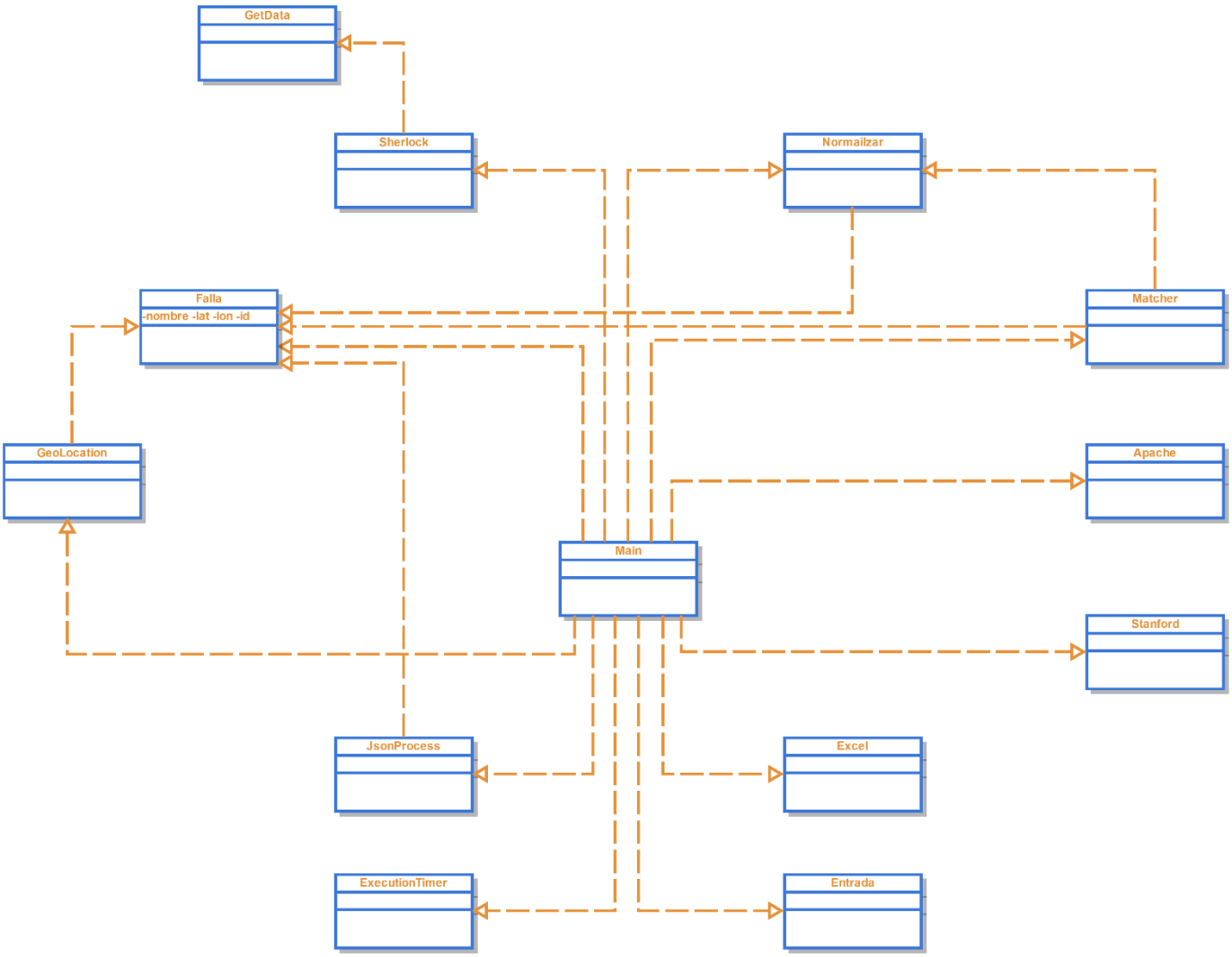

Figura 3.13 Relación entre las clases del paquete NER

3.4.2. Ciclo de vida

En esta sección se pretende describir el proceso secuencial experimentado en la ejecución de la herramienta. Al girar todo el sistema alrededor de una estructura switch de control de flujo implementada en la clase principal 'Main' del paquete NER, el proceso secuencial se bifurca y puede tomar distintos caminos.

La primera labor de la herramienta es la de procesar y cargar la información provenienente del fichero JSON de opendatavalencia. El segundo paso de la herramienta es la de cargar las noticias almacenadas localmente en el directorio "Noticias". El siguiente paso es el correspondiente al módulo NER Set-Up y cuya misión es la de instanciar la clase Matcher – funciones LingPipe – ya que los tres módulos NER

implementan LingPipe[4]. Tras instanciar Matcher, mediante un menú switch se elige la función deseada.

Las distintas funciones a elegir pueden ser:

[1] Lector RSS: Extraer noticias nuevas de los feeds RSS

[2] Apache.Trainer: Crear un modelo Apache NER

[3] NER Logistics: Crear un corpus Stanford a partir de corpus Apache

[4] Estudio Comparativo: realizar un estudio comparativo de los tres métodos NER

[5] LingPipe: Analizar noticias mediante herramienta NER LingPipe y opcionalmente geolocalizar las fallas mencionadas identificadas.

[6] Apache: Analizar noticias mediante herramienta NER Apache Open NLP y opcionalmente geolocalizar las fallas mencionadas identificadas.

[7] Stanford: Analizar noticias mediante herramienta NER Stanford y opcionalmente geolocalizar las fallas mencionadas identificadas.

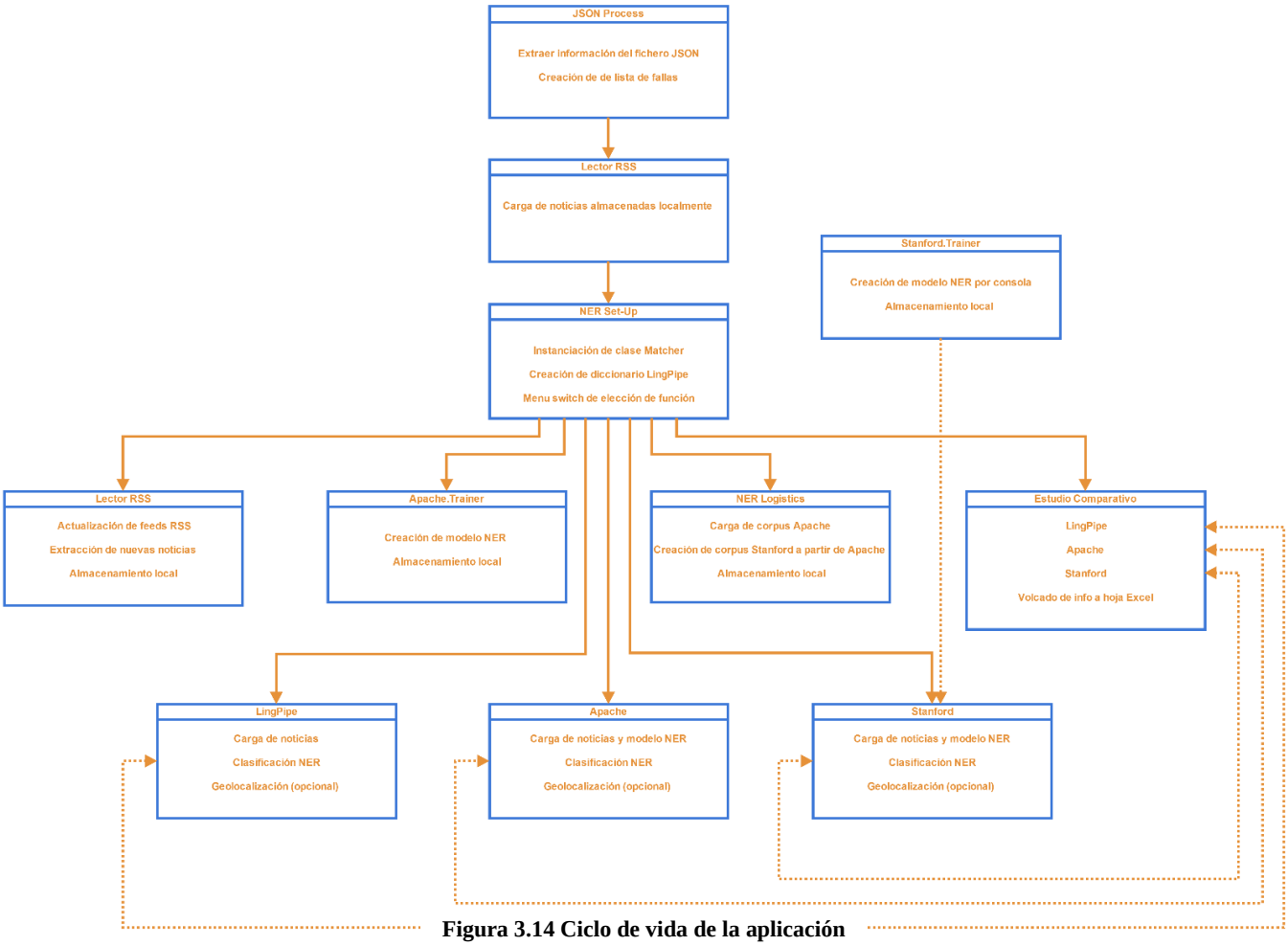

Figura 3.14 Ciclo de vida de la aplicación

[4] Módulo ampliamente detallado en el apartado 3.4.6

3.4.3. Módulo lector RSS

El módulo "lector RSS" es el inicio de todo. La génesis de la herramienta. La fuente de información. Uno de los requisitos indispensables en la ejecución del proyecto era la utilización de los canales RSS de divulgación como fuente de información desde donde nutrirse de noticias.

En esencia, este módulo está compuesto de un lector de RSS, también conocido como agregador, y una salida dirigida a un directorio local donde transferir las noticias pertenecientes a los feeds RSS.

El lector RSS es un tipo de software incorporado en la estructura del proyecto para suscribirse a fuentes de noticias en formato RSS, Atom y otros derivados de XML. El agregador reúne noticias publicadas en los feeds seleccionados. La implementación del lector de RSS en el sistema tiene como fuente un código open-source, posteriormente refactorizado y ampliado con el fin de satisfacer las necesidades de la herramienta.

La base del código open-source del que se hace uso para el lector de RSS ha sido retocado en pequeños detalles, a los que hay que añadir las nuevas funcionalidades aplicadas. El lector carga en primer lugar una serie de feeds de los que leer noticias publicadas. Posteriormente, mediante el uso de una clase FeedMessage, se vuelcan las noticias a la memoria del programa. Posteriormente, el programa accede al directorio local "Noticias" donde coteja las noticias existentes con las nuevas y en caso de no ser una repetida ejecuta dos operaciones. Almacena la noticia en un nuevo fichero y vuelca el contenido de la noticia a un fichero llamado "Anexo.txt" sobre el cual se añade por frases cada noticia nueva.

El objetivo de este fichero "Anexo.txt" es el de volcar todas las noticias para posteriormente crear un corpus, elemento imprescindible para la creación de un modelo de clasificación NER tanto en Stanford NLP como en Apache Open NLP. A su vez, el fichero Anexo.txt era revisado para eliminar frases redundantes. Es decir, exactamente iguales.

En cuanto a la selección de feeds se valoró una lista de aproximadamente veinte. En todos ellos se publicaba información referida a las Fallas. No obstante, siguiendo recomendaciones en lo concerniente a la creación de modelos de clasificación NER, se descartaron aquellos cuya temática principal no eran las Fallas. Por ejemplo, aquellos

feeds cuya temática son eventos relacionados con la ciudad de Valencia en general. Este tipo de feeds suponían un problema en la creación de un corpus para generar un modelo de clasificación NER, ya que el corpus debe tener información relativa a la temática cuya categoría ha de ser clasificada. La información inconexa deriva en aumentar el tamaño del fichero y el tiempo de ejecución del modelo a posteriori.

Por ello, la selección final de feeds consta de únicamente siete:

[1] http://www.20minutos.es/rss/minuteca/fallas/"
[2] http://www.actualidadfallera.es/es/noticias-fallas-de-valencia?format=feed&type=rss"
[3] http://estaticos01.elmundo.es/rss/espana/fallas-de-valencia.xml
[4] http://www.hablemosdefallas.es/?format=feed&type=rss
[5] http://www.levante-emv.com/elementosInt/rss/63
[6] http://elpais.com/tag/rss/fallas/a
[7] http://www.lasprovincias.es/rss/2.0/?seccion=fallas

Por último, es preciso destacar que este módulo toma de nuevo un papel destacado a la hora de clasificar noticias, ya sea mediante la ejecución de una de las tres herramientas NER de forma aislada o al realizar un estudio comparativo, debido a que este módulo es el encargado de cargar las noticias almacenadas localmente en el directorio "Noticias".

3.4.4. Módulo JSON

Debido al hecho de enfocar la temática del presente proyecto a las Fallas de Valencia 2015, surge la imprescindible necesidad de contar con una relación de todos los monumentos falleros participantes. El Ayuntamiento de Valencia, a través de su web en la sección Datos Abiertos ofrece un fichero en formato JSON en el que se reúne toda la información concerniente a los monumentos falleros.

En concreto, cada monumento fallero consta de las siguientes propiedades:

- **Id**
- **Nombre**
- **Sección**
- **Fallera mayor**
- **Presidente**
- **Artista**
- **Lema**
- **Situación geográfica**

Este fichero JSON es el punto de partida del módulo denominado también como JSON, cuya complejidad es reducida. La implementación de este módulo se reduce únicamente al proceso de extracción del archivo JSON.

Con esta finalidad, se crea la clase JsonProcess en el paquete NER. Esta clase contiene el método estático getFallas, donde se desarrolla todo el proceso de manipulación y extracción. El método getFallas extrae la información del fichero JSON, el cual está almacenado localmente, en codificación UTF-8 – para cumplir posteriormente con los requisitos de codificación de los módulos NER – volcando la información en un ArrayList y apoyándose en la clase Falla, cuyos atributos pasan a ser las propiedades del fichero JSON.

Además, se filtra la información haciendo uso del método estático NormalizarJSON dentro de la clase Normalizar. La clase Normalizar tiene funciones de depuración de cadenas. Consta de dos métodos: NormalizarJSON y NormalizarFrase con pequeñas variaciones entre ambas ya que, como sugieren sus nombres, sus actividades están más enfocadas en la depuración por un lado del fichero JSON y por otro lado de NormalizarFrase.

La correcta manipulación del fichero y su posterior proceso de depuración es clave a posteriori en labores de Reconocimiento de Nombres de Entidades. Una manipulación simple a modo de acondicionamiento de los nombres de los monumentos falleros aumenta las capacidades de los métodos NER de reconocer entidades. No obstante, se hará énfasis en este sentido en la descripción de la implementación del módulo motor NER.

3.4.5. Módulo NER Logistics

El proceso de clasificación NER no es inmediato. Conlleva una preparación enfocada a los objetivos y una elaboración previamente meditada. En los tres métodos de NLP desarrollados en el presente proyecto – LingPipe, Apache Open NLP y Stanford NLP – se pretende, obviamente, identificar el mayor número posible de menciones de monumentos falleros. LingPipe se apoya en un diccionario y es por tanto un proceso semiautomático, pero ¿Qué ocurre con Apache y Stanford?

Tanto Apache Open NLP como Stanford NLP basan sus capacidades y funcionalidad en un modelo de clasificación en su modalidad de reconocimiento de nombres de entidades (NER). Tal y cómo se explica en los apartados 2.2.2 Apache Open NLP y 2.2.3 Stanford NLP , los modelos de clasificación se generar a partir de la creación de un corpus siguiendo el formato especificado por cada desarrollador para cada servicio NER. El corpus está compuesto de texto relacionado con la categoría a clasificar.

Es decir, nuestros corpuses – en inglés *corpora* – van a estar compuestos de noticias relacionadas con las Fallas.

En el caso de los dos servicios NER utilizados basados en modelos de clasificación, el corpus debe ser elaborado 'a mano'. La razón es sencilla: se ha de 'entrenar' al corpus. Es decir, una vez obtenemos todos los requisitos para la elaboración del corpus, siguiendo el formato de etiquetación de menciones especificado para cada servicio debemos 'marcar' cuándo existe una mención que nosotros desearemos identificar en un futuro haciendo uso del modelo.

En el caso de Apache, las recomendaciones en cuanto a extensión son las de crear un corpus con al menos 15.000 líneas de texto y, para el caso de Stanford, 45.000 tokens.

Debido a la gran laboriosidad de esta tediosa tarea y a su longtiud, se decide crear una arquitectura capaz de generar un corpus a partir de otro y no realizar labores de forma redundante.

Para ello, se crea la siguiente arquitectura, representativa del módulo NER Logistics:

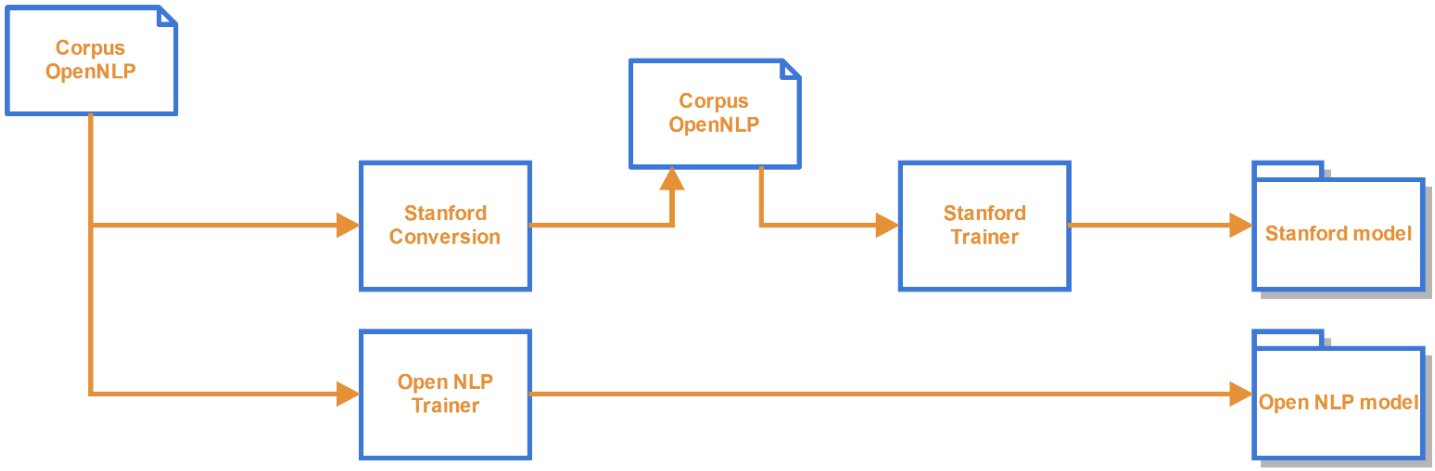

Figura 3.15 Arquitectura de módulo NER Logistics

En primer lugar, se recuerda que el corpus de Apache Open NLP es el fichero Anexo.txt generado en el módulo Lector RSS. Este fichero, posteriormente es depurado, revisado y etiquetado 'a mano' aplicándole así el formato especificado por Apache. Una vez obtenido el corpus de Apache, observamos en Figura 3.15 Arquitectura de módulo NER Logistics que mediante el uso de la clase StanfordConversion y su método estático

convert generamos un corpus con las características exigidas por Stanford. Es decir, transformamos del corpus representado a la izquierda de la tabla Tabla 3-1 al representado a su derecha.

La joven Estefanía López Montesinos de la comisión <START:falla> Embarcadero Historiador Beti <END> y la niña María Donderis Sanchis de la Falla <START:falla> Blasco Ibáñez Plaza Maestro Ripoll <END> son las nuevas Fallera Mayor y Fallera Mayor Infantil de Valencia del año 2015 según ha anunciado la alcaldesa de la ciudad Rita Barberá representante de la comisión <START:falla> Horta Sud La Costera <END>	la O comisión O Embarcadero FALLA Historiador FALLA Beti FALLA y O la O niña O María O Donderis O

Tabla 3-1Formatos Open NLP vs Stanford

A partir del corpus de Apache, se observa que no sólo existe la posibilidad de crear un corpus para Stanford, sino que también podemos crear el modelo de clasificación de Apache. Ambas opciones están disponibles en el menú switch de selección de la clase principal Main en el paquete NER del software.

A partir del corpus de Apache, se observa que no sólo existe la posibilidad de crear un corpus para Stanford, sino que también podemos crear el modelo de clasificación de Apache. Ambas opciones están disponibles en el menú switch de selección de la clase principal Main en el paquete NER del software.

También perteneciente al paquete NER, se encuentra la clase Apache. Apache es una clase con dos métodos: Trainer, para la creación del modelo de clasificación NER; y Spotlight, con funciones de clasificación.

En el caso de seleccionar la opción de entrenar un modelo Apache en el menú switch de la clase Main, se ofrecen opciones para elegir qué corpus se desea elegir para la creación del modelo, en caso de haber más uno. Una vez elegido, se ejecuta la llamada

al método estático Trainer mencionado anteriormente. También se ofrece la opción de elegir un nombre para el modelo a crear.

El funcionamiento interno de este método de tipo 'booleano' tiene una serie de parámetros configurables para la óptima creación del modelo:

- **Cutoff**: Especifica el mínimo número de veces que una categoría debe aparecer en el corpus para que el modelo adopte funcionalidades de clasificación para ella.
- **Codificación**: Determina el tipo de codificación empleada.
- **Iterations**: Determina el número de veces que se entrena o repasa el corpus en el entrenamiento del modelo.
- **Lenguaje**: Especifica el lenguaje empleado en el corpus. Se puede especificar más de uno.
- **Categorías**: Categorías a identificar con el modelo y referenciadas en el corpus.
- '**tokenizador**' a cargar: En el caso de creación de modelos NER, para entrenar un corpus, se ha de hacer uso de tokenizador que siga las especificaciones también de Apache Open NLP. Para ello hemos utilizado los modelos propios de tokenización predefinidos por Apache.
- **Corpus** a cargar: Hace referencia al corpus que se desea utilizar y sobre el que entrenar el modelo.

En caso de creación satisfactoria del modelo, el método devolverá el valor booleano 'true' y almacenará localmente el modelo en la carpeta "…\Trainers\Apache". En este proyecto se han realizado distintas pruebas modificando parámetros en búsqueda de la optimización de los modelos utilizados. Con este objetivo, se ha jugado con el número de iteraciones y el tokenizador.

Se adjunta a continuación una tabla haciendo referencia a la modificación del número de iteraciones y a su repercusión en el modelo:

MODELO	A	B	C	D
Número de Iteraciones	80	100	120	250
Tamaño (kB)	833	833	833	834
Tiempo de ejecución medio (ms)	185	175	182	186
Fallas identificadas	105	106	106	106

Tabla 3-2 Evolución de modificación del parámetro 'Iteraciones'

Utilizando como argumento principal la lectura de la Tabla 3-2 en el que se aprecia que tanto el menor tiempo de ejecución medio como el máximo número de fallas identificadas pertenecen al modelo creado con 100 iteraciones, se decide implementar todos los modelos sobre 100 iteraciones en el entrenamiento.

Una vez obtenido un modelo para Apache Open NLP y generado un corpus para Stanford NLP, igualmente podemos crear o entrenar un modelo de clasificación NER para Stanford. No obstante, varían los procedimientos.

En el caso de Apache, la creación del modelo se realiza mediante la API suministrada por Apache. Sin embargo, Stanford dispone de un manejo por consola muy sencillo por lo que se crea por consola, tal y como se detalla en el punto 2.2.3 Stanford NLP.

Una vez descritas las diferentes funciones que desempeña el módulo NER Logistics, es fácil comprender el porqué de la relación indirecta de este módulo con el resto del diseño. La razón es que no existe relación directa en sí en la implementación de la herramienta, pero sí a través de los ficheros generados entre ellas y almacenados localmente.

3.4.6. Módulo motor NER

Generalmente, las aplicaciones de software que implementan herramientas de Reconocimiento de Nombres de Entidades (NER) utilizan éstas como objetivo final. Es decir, el objetivo final de estas aplicaciones es la de mostrar las entidades reconocidas. Son aplicaciones de estas características, por ejemplo, las aplicaciones mencionadas en el apartado 1.2 Estado del arte donde o bien se hacen búsquedas de puntos geográficos de forma automática o bien simplemente se muestran las menciones y, también con la misma finalidad, la GUI para NER suministrada por Stanford NLP en su página web.

Sin embargo, en el caso concreto de este proyecto, las herramientas NER no suponen un fin sino un medio. Reconocer nombres de monumentos falleros es un paso intermedio cuya meta final es la de geolocalizar aquellas fallas cuyas menciones han sido localizadas. Es decir, manejar la salida de las fallas identificadas se convierte en un paso primordial ya que éstas deben ser geolocalizadas a posteriori.

Por consiguiente, se decide englobar el uso de los servicios Apache y Stanford con LingPipe. La salida de LingPipe viene determinada en función de las entradas del

diccionario que implementa y de esta forma el margen de ocurrencia de falsos positivos se reduce considerablemente. En cambio, Apache y Stanford están basadas en modelos de clasificación, por lo que su salida no es tan predecible. Y no sólo eso. En el caso ideal de disponer de un modelo de clasificación de monumentos falleros con una precisión del 100% no podríamos garantizar que todas las fallas fueran de la ciudad de Valencia. Ésta es, por tanto, una razón más para garantizar que la salida generada por Apache y Stanford se reduzca a aquellos monumentos falleros dentro de nuestro espectro de identificación.

Partiendo de esta premisa surge la creación del módulo denominado motor NER. La implementación del mencionado módulo se reduce únicamente a la instanciación de la clase Matcher, es decir, el servicio NER de LingPipe. La instanciación de la clase Matcher requiere del paso del parámetro listaFallas – un ArrayList de clase Falla derivado del módulo JSON – en su constructor. La clase Matcher en su instanciación simplemente crea el diccionario en el que incorpora los nombres de los 350 monumentos falleros que aparecen en el fichero JSON. Además, en la creación del diccionario se aborda una de las trabas presentadas a lo largo del desarrollo del proyecto. Los nombres de los monumentos falleros de la ciudad de Valencia poseen ciertas características que es necesario abordar:

- **Bilingüismo**: En las noticias obtenidas en los feeds RSS se observa que un gran número de menciones de fallas mantienen sus menciones en valenciano a pesar de tratarse de noticias en castellano.
- **Seudónimos**: Si bien se trata de un número reducido de fallas, su cuota de menciones es elevada al tratarse de fallas pertenecientes a categorías importantes como Especial o 1A y 1B.
- **Cuota de menciones**: La distribución de atención que reciben los 350 monumentos falleros no es ni mucho menos igualitaria. Es decir, la atención mediática a la que se expone una falla guarda una relación directa con la sección a la que pertenece.

Se adjuntan a continuación tablas donde se sustentan estas características:

Nombre de falla	Valenciano
Quart Extramurs - Velazquez	Quart Extramurs - Velazquez
Plaza del Negrito	Plaza del Negret
Regne de Valencia - Duque de Calabria	Regne de Valencia - Duc de Calabria
Duque de Gaeta - Puebla de Farnals	Duc de Gaeta - Pobla de Farnals
Maestro Gozalbo - Conde Altea	Mestre Gozalbo - Compte Altea

Tabla 3-5 Fallas con menciones recurrentes en valenciano

Nombre de falla	Seudónimo
Ribera - Convento Santa Clara	Telefónica
General Llorens - Dr. M. Merenciano	Els generals
Dr. Gil y Morte - Dr. Vila Barberà	Els doctors
Jorge Comin - Serra Calderona	Nou Campanar

Tabla 3-4 Fallas con menciones frecuentes bajo un alias

Sección	Cuota	Cuota(%)
Especial	75,00	36,41%
1A	37,00	17,96%
1B	21,00	10,19%
2A	8,00	3,88%
2B	18,00	8,74%
3A	9,00	4,37%
3B	16,00	7,77%
Resto	22,00	10,68%
Total	206,00	100,00%

Tabla 3-3 Cuota de atención mediática por categorías

Teniendo en cuenta estas singularidades, se han incorporado entradas dedicadas de forma extraordinaria en el diccionario para que aquellas fallas a las que se alude de forma frecuente tanto en valenciano como en castellano, mediante el uso de algún sobrenombre tengan un tratamiento más dedicado. Igualmente, y debido a la cuota de atención mediática recibida en función de la sección a la que pertenecen los monumentos falleros, las fallas pertenecientes a las categorías *Especial*, *1A* y *1B* con una cuota conjunta del 64,56% reciben un trato prioritario. Las fallas pertenecientes a estas tres categorías reciben un trato especial. Más concreto. Por ello, tienen todas ellas entradas dedicadas en el diccionario cubriendo varias condiciones especiales observadas en artículos de

noticias, tales como abreviaciones, uso de iniciales, etcétera, con el fin de aumentar la precisión en las secciones que más atención reciben.

No obstante, las entradas detalladas en el diccionario no sólo sirven para garantizar que la mención trata en efecto de un monumento fallero en los casos de uso de Apache Open NLP y Stanford NLP, sino que también aumenta la precisión de LingPipe, ya que al no estar basado éste en modelos de clasificación, sus capacidades están basadas en la fuerza que tenga su base. Es decir, el diccionario entre otros parámetros.

Por último, es necesario destacar el uso de la clase Normalizar en el paquete NER, ya que su misión es la de filtrar y modificar palabras clave y, de esta forma, mantener un filtro único tanto para la denominación de los monumentos falleros provenientes del fichero JSON como para las menciones halladas en los flujos RSS.

La clase Normalizar desarrolla dos métodos estáticos. Uno para JSON, comentado en el punto 3.4.4 y otro que normaliza por frases. La decisión de implementar la entrada del método en frases y no en noticias completas reside en que se realiza en un punto de la implementación donde ya existe un bucle y realizarlo previamente provocaría el uso de otro bucle aparte, aumentando de esta forma el tiempo de ejecución. Las funciones del método estático normalizarFrase son las de eliminar caracteres especiales, tildes y la eliminación o modificación de palabras clave. Esta clase tiene la misión de "normalizar" el uso de palabras recurrentes en los nombres de las fallas, i.e.: Plaza, Avenida. En función de la palabra, o bien elimina cualquier versión de la palabra o bien la modifica a modo de unificación. Es decir, si en una frase apareciese la abreviación "PZ.", el método la modificaría a "PLAZA" y así facilitar las labores de clasificación a posteriori.

3.4.7. Módulo LingPipe

LingPipe es uno de los tres servicios NER propuestos para el reconocimiento de nombres de monumentos falleros en flujos RSS. A diferencia de los otros dos servicios (Apache Open NLP y Stanford NLP), el funcionamiento de LingPipe no se basa en el uso de un modelo de clasificación de entidades. Es por ello que su implementación es bien distinta.

Ya se ha mencionado en el apartado anterior (3.4.6 Módulo motor NER) que la utilización del método NER de LingPipe se subdivide en dos partes:

- **Control de calidad**: Se utiliza en primer lugar como parte del módulo motor NER, donde la clase Matcher del paquete NER se instancia. Es en el momento de la instanciación cuando se genera el diccionario. Base de la utilización a posteriori de la clasificación NER. La razón de esta implementación se debe a que como control de calidad se utiliza la clasificación LingPipe sobre las menciones halladas por Apache Open NLP y Stanford.

  En el caso de utilizar las herramientas de clasificación Apache Open NLP o Stanford, en primer lugar se utilizan estos métodos, los cuales devuelven un vector de las menciones que pasa a ser analizado por el método de clasificación LingPipe. Finalmente, es LingPipe el que confirma o descarta la mención. Es decir, las herramientas Apache y Stanford no son puras, sino que también se hace uso de LingPipe a modo de confirmación.

- **Herramienta NER**: La herramienta de clasificación NER tiene lugar mediante la llamada al método *spotlight* de la clase Matcher. Esta llamada se realiza por elección en el menú switch de la clase principal Main en el paquete NER.

  En primer lugar, como paso previo a la llamada del método, ya en la instanciación se determinan los parámetros utilizados para determinar la búsqueda. A pesar de haberse descrito ya la instanciación de la clase en el apartado 3.4.6, en este apartado se detalla la elección y el proceso de selección de los parámetros que determinan el funcionamiento de LingPipe.

  Los parámetros a determinar son los que han de ser pasados en el constructor de la clase chunker. Esta clase es de instanciación obligatoria y su uso en la implementación de este proyecto sigue las recomendaciones provistas en la propia web de Alias-I. Los parámetros pasados en el constructor son:

  - Diccionario
  - Clase TokenizerFactory
  - Clase EditDistance
  - Parámetro maxDistance

  La clase TokenizerFactory contiene a su vez otros parámetros instanciados como clases, como la clase WhiteSpaceNormTokenizerFactory e IndoEuropeanTokenizerFactory. El conjunto de esta selección detallada a

continuación determina que se convertirán todos los espacios de longitud mayor a uno en uno solo y posteriormente se filtrarán en la generación de tokens, Es decir, se "ignorarán" los espacios en blanco. Además, se especifica que se utiliza un idioma latino o angosajón, no utilizar sensibilidad entre mayúsculas y minúsculas y que en caso de identificar una mención, no seguir buscando más combinaciones para hallar otra mención con los mismos tokens. Estas dos últimas opciones son las determinadas por defecto marcando el atributo INSTANCE aunque son modificables.

```
TokenizerFactory tokenizerFactory = new
WhitespaceNormTokenizerFactory(IndoEuropeanTokenizerFactory.INSTANCE)
```

Los atributos más importantes sin embargo son los cinco parámetros incluidos en la instanciación de la clase editDistance y el parámetro maxDistance. Un total de seis parámetros que actúan como pesos ponderados y ya descritos en el apartado 2.2.1LingPipe:

- **match**: Peso otorgado a un chunk cuya coincidencia es exacta.
- **delete**: Peso otorgado a la eliminación de un carácter para buscar la coincidencia.
- **insert**: Peso acreditado a la inserción de un carácter.
- **substitute**: Peso asignado en caso de sustituir un carácter en búsqueda de coincidencias.
- **transpose**: Peso acreditado a la transposición entre caracteres contiguos.
- **maxDistance**: Límite máximo establecido de aproximación.

Para determinar los parámetros óptimos para el reconocimiento de nombres de monumentos falleros, se ha llevado a cabo un estudio de comparación y efectividad de parámetros utilizando la herramienta LingPipe sobre siete noticias distintas. Los criterios de selección de parámetros tal y como se observa en las tablas mostradas a continuación se basan en el tiempo de ejecución de LingPipe sobre la noticia y el número de fallas hallado. En la columna NUM FALLAS aparece entre paréntesis un número, el cual hace referencia al número real de menciones de fallas presentes en la noticia.

News1	MATCH	DELETE	INSERT	SUBS	UMBRAL	FALSE POS.	TIEMPO EXE (ms)	NUM FALLAS (26)
	0	-2	-2	-1	4	3	5684	26
	0	-2	-2	-2	4	0	1566	24
	0	-2	-2	-3	5	0	1232	24
	0	-2	-3	-1	4	3	3311	26
	0	-1	-2	-2	5	0	3372	24
	0	-3	-3	-2	6	0	2151	24
	0	-2	-2	-1	5	12	17836	38
	0	-2	-3	-2	5	0	1184	24
	0	-2	-3	-2	4	0	714	24
	0	-1	-3	-3	4	0	525	24
	0	-3	-3	-2	6	0	2126	24
	0	-3	-3	-2	4	0	573	24
	0	-1	-2	-2	4	0	1653	24

Tabla 3-8 Estudio de parámetros 1

Los conjuntos de parámetros que van dando malos resultados van siendo eliminados y los que muestran resultados aceptables son puestos a prueba con la siguiente noticia. De esta forma, se reduce el abanico de posibilidades.

News2	MATCH	DELETE	INSERT	SUBS	UMBRAL	FALSE POS.	TIEMPO EXE (ms)	NUM FALLAS (36)
	0	-2	-2	-2	4	0	1779	33
	0	-2	-2	-3	5	0	1508	33
	0	-2	-3	-2	5	0	1486	33
	0	-2	-3	-2	4	0	782	32
	0	-1	-3	-3	4	0	633	33
	0	-3	-3	-2	4	0	628	32
	0	-1	-2	-2	4	0	2111	33

Tabla 3-6 Estudio de parámetros 2

News3	MATCH	DELETE	INSERT	SUBS	UMBRAL	FALSE POS.	TIEMPO EXE (ms)	NUM FALLAS (10)
	0	-2	-2	-2	4	0	2577	10
	0	-2	-2	-3	5	0	2004	10
	0	-2	-3	-2	5	0	2030	10
	0	-1	-3	-3	4	0	741	9
	0	-1	-2	-2	4	0	2917	10

Tabla 3-7 Estudio de parámetros 3

News4	MATCH	DELETE	INSERT	SUBS	UMBRAL	FALSE POS.	TIEMPO EXE (ms)	NUM FALLAS (10)
	0	-2	-2	-2	4		2370	10
	0	-2	-2	-3	5		1822	10
	0	-2	-3	-2	5		1797	10
	0	-1	-3	-3	4		727	9

Tabla 3-12 Estudio de parámetros 4

News5	MATCH	DELETE	INSERT	SUBS	UMBRAL	FALSE POS.	TIEMPO EXE (ms)	NUM FALLAS (1)
	0	-2	-2	-3	5		334	1
	0	-2	-3	-2	5		322	1
	0	-1	-3	-3	4		204	1

Tabla 3-11 Estudio de parámetros 5

News6	MATCH	DELETE	INSERT	SUBS	UMBRAL	FALSE POS.	TIEMPO EXE (ms)	NUM FALLAS ()
	0	-2	-2	-3	5	0	2178	12
	0	-2	-3	-2	5	0	2109	12
	0	-1	-3	-3	4	0	905	11

Tabla 3-10 Estudio de parámetros 6

News7	MATCH	DELETE	INSERT	SUBS	UMBRAL	FALSE POS.	TIEMPO EXE (ms)	NUM FALLAS ()
	0	-2	-2	-3	5	0	1507	32
	0	-2	-3	-2	5	0	1598	32
	0	-1	-3	-3	4	0	749	32
	0	-3	-3	-1	4	2	3048	33

Tabla 3-9 Estudio de parámetros 7

Basado en este estudio se decide establecer el conjunto [0,-1,-3,-3] y 4 debido al alto porcentaje de reconocimiento en un tiempo de ejecución notablemente reducido respecto al resto.

Ya en lo referente a la implementación del método *spotlight*, su primera tarea es dividir la noticia pasada como parámetro por frases. Después, cada frases es depurada haciendo uso del método estático normalizarFrase de la clase Normalizar y posteriormente pasar a hacer “chunking”, almacenando también la secuencia de la frase y la disposición entre los distintos tokens formando grupos por frases “ChunkSet”.

Cada ChunkSet es analizado exhaustivamente y es precisamente en este análisis donde tiene lugar el reconocimiento de entidades. Se hace uso del diccionario y de los parámetros establecidos y LingPipe hace las operaciones necesarias iterativamente hasta determinar si en el ChunkSet existe alguna mención de las localizadas en el diccionario. Si es así, LingPipe devuelve la mención en forma de clase Falla y a su vez ésta se almacena en un HashMap cuya clave par es su propia id. El servicio NER de LingPipe ofrece la posibilidad de comprobar la distancia con la que se ha encontrado la mención, la cadena original (ChunkSet en crudo) y la cadena final con las transformaciones realizadas. Finalmente, el método *spotlight* devuelve el HashMap con la relación de todas las fallas identificadas.

En referencia a la implementación dentro de la opción correspondiente del menú switch en la clase Main del paquete NER, el código es más extendido. En primer lugar, al acceder al 'caso LingPipe' se crea un bucle que itera las distintas noticias contenidas en la lista de noticias cargada en memoria a partir de las noticias almacenadas localmente. A continuación, dentro del bucle, se instancia la clase ExecutionTimer, una clase dedicada a la medición de tiempos de ejecución de índole open-source y entre el inicio y el cierre de la medición, se ejecuta el método spotlight. No obstante, todas las funciones de medición de tiempos implementadas en el proyecto sirven como análisis interno y estadístico, no es que supongan una mejora determinante en el resultado final de la aplicación.

3.4.8. Módulo Apache Open NLP

Apache Open NLP es el segundo de los tres servicios NER implementados para la identificación de monumentos falleros en flujos RSS. A modo de resumen, se repasa en este apartado el funcionamiento de Apache como recordatorio:

[1] Creación de **corpus** en formato especificado y extensión .TRAIN

[2] Entrenamiento de **modelo NER**, generando archivo .bin

[3] Utilización de modelo para **reconocer nombres** de las entidades para las cuales se ha entrenado el modelo.

Los pasos [1] y [2] ya han sido descritos previamente en el punto 3.4.5, por lo que en este apartado se procede a describir únicamente el funcionamiento e implementación del último punto.

Para acometer el reconocimiento de entidades de monumentos falleros en noticias mediante el uso del servicio NER de Apache, se crea la clase homónima, Apache, perteneciente al paquete NER. Esta clase ha sido desarrollada gracias a la API proporcionada por Apache y mediante la utilización también del modelo creado en el módulo NER Logistics.

La clase Apache, consta de dos métodos estáticos: *trainer*, ya descrito en el punto 3.4.5, y *spotlight*, centrado en la clasificación de entidades y donde se implementa la herramienta NER.

En primer lugar, la llamada al método spotlight, se realiza en la opción correspondiente del menú switch de la clase Main, donde se le pasa como parámetro el artículo de una noticia en un único String o cadena de texto.

Una vez, dentro del método en sí, el primer paso es la carga de un archivo *tokenizador* .bin. En este proyecto se ha utilizado uno provisto por Apache y cuyo funcionamiento ha demostrado ser de garantías. A continuación, el tokenizador convierte la totalidad del artículo en tokens para después pasar al proceso de identificación de nombres. Es aquí, en este paso donde se carga el modelo de clasificación previamente creado. Una vez detectados las posibles menciones, cada una de éstas es cotejada mediante procesos probabilísticos y a cada mención le es asignada una probabilidad. Basándose en esta probabilidad, Apache decide descartarla o no. Finalmente, cada mención que pasa el corte es almacenada en un ArrayList y posteriormente devuelta (*return*).

Respecto a la implementación de la opción de clasificación NER con Apache dentro del menú switch de la clase Main, el método spotlight viene envuelto por la utilización de LingPipe a modo de control de calidad. La opción ‘4’ del menú, correspondiente a Apache, comienza con la instanciación de la clase ExecutionTimer, una clase open-source de medición de tiempos la cual es empleada para determinar el tiempo de ejecución del reconocimiento de entidades de Apache. Inmediatamente a continuación, se ejecuta la clasificación para cada artículo dentro de un bucle de iteración de toda la lista disponible de noticias. Dentro del bucle, en primera instancia se ejecuta la

herramienta Apache, haciendo uso del método spotlight. El método devuelve una lista con las fallas identificadas y éstas, a su vez, pasan por un proceso de control de calidad haciendo uso de LingPipe. LingPipe analiza las menciones y devuelve la lista de menciones definitiva y pasa esta lista a la clase GeoLocation, de ejecución opcional con el fin de posicionar geográficamente las menciones identificadas.

3.4.9. Módulo Stanford NLP

Stanford NLP es el último de los tres servicios NER propuestos en la búsqueda de la herramienta NER óptima para reconocer monumentos falleros. El funcionamiento de Stanford NER es similar al del Apache NER. El proceso de Stanford se basa en los tres mismos pasos:

[1] Creación de **corpus** en formato especificado y extensión .TSV y fichero de configuración .prop

[2] Entrenamiento de **modelo NER** de extensión **.ser.gz**, a través de consola.

[3] Utilización de modelo para **reconocer nombres** de las entidades para las cuales se ha entrenado el modelo.

Las labores de reconocimiento de nombres son implementadas dentro de la clase Stanford mediante el método estático *spotlight*.[5]

El proceso de reconocimiento de nombres de Stanford comienza mediante la llamada al método spotlight de Stanford el cual, al igual que Apache, recibe el artículo de una noticia en un único String como parámetro. Una vez dentro del método se leen los argumentos pasados por referencia en consola y se carga el modelo de clasificación NER almacenado en el directorio "…`/Trainers/Stanford/`".

Posteriormente, Stanford separa en líneas el artículo y de forma iterativa las analiza. El análisis por líneas se basa en la utilización del método estático classifyToString perteneciente a la clase classiffier. Este método es facilitado por la API

[5] Aclaración: A pesar de que las tres herramientas NER (LingPipe, Apache y Stanford) implementan un método de clasificación NER denominado spotlight, se trata de tres métodos distintos, cada uno perteneciente a sus respectivas clases.

de Stanford y se ejecuta con los parámetros “tsv” y “false” tal y como se observa a continuación:

```
classifier.classifyToString(str, "tsv", false)
```

Esta orden ejecuta la clasificación apoyada en determinación mediante procesos heurísticos y donde tsv apunta que en el texto de clasificación las categorías están separadas por columnas (Tab Separated Values) y el valor booleano a continuación determina si preservar el espacio en blanco entre tokens el cual podría ser nulo en ocasiones (true) o si "tokenizar” el texto e imprimirlo con un espacio entre cada token (false). Por último, dentro del bucle se realiza el último paso. Si las menciones identificadas en la línea no se han detectado previamente se añaden a una lista de menciones detectadas (listaMenciones) y devuelta mediante return a la llamada del método.

La llamada del método se realiza mediante la elección de su opción correspondiente en el menú switch de la clase Main del paquete NER. La estructura de la opción Stanford en el switch es idéntica a la implementada en Apache a excepción del uso de Stanford por Apache. Se envuelve el proceso mediante la medición de tiempos con la clase ExecutionTimer y se itera en un bucle sobre todas las noticias disponibles, artículo por artículo. Dentro del bucle se ejecuta el método *spotlight* el cual devuelve la lista de menciones detectadas e inmediatamente después, se analizan las menciones a modo de control de calidad mediante LingPipe, el cual ya devuelve la lista definitiva de fallas localizadas a geolocalizar. La geolocalización, al igual que en las opciones de clasificación LingPipe y Apache, también es opcional.

3.4.10. Módulo Estudio Comparativo

El objetivo de este proyecto no trata únicamente de geolocalizar las entidades reconocidas en noticias sino que también persigue encontrar el método NER más adecuado para reconocer menciones de nombres de monumentos falleros.

Con el objetivo de determinar cuál de los tres métodos NER es el más adecuado, se decide implementar el módulo Estudio Comparativo. Este módulo es una opción dentro del menú switch de la función Main del paquete NER. La finalidad del módulo es la realización de un estudio analítico comparativo de los tres métodos NER sobre el que basar la decisión.

La implementación de este método se basa principalmente en la ejecución secuencial de las tres herramientas NER implementadas y descritas anteriormente. Debido al manejo de ficheros almacenados localmente, el proceso es secuencial y no paralelo, como bien podría haberse diseñado en el caso de uso de una base de datos relacional. El inconveniente reside en el riesgo de lectura y escritura simultánea sobre noticias y ficheros.

El funcionamiento por tanto, consiste en el análisis de las noticias recogidas de los flujos RSS de forma que, en primer lugar, se ejecuta la herramienta LingPipe, a continuación Apache y por último Stanford. La ejecución de sendas herramientas es idéntica a las ejecutadas anteriormente de forma aislada, a excepción de que no se implementa geolocalización.

En la secuencia se lleva a cabo un control exhaustivo de los tiempos de ejecución y menciones identificadas por noticia con cada uno de los tres métodos.

Finalmente, como último paso, se vuelca toda la información en una hoja de cálculo Microsoft Excel. Para ello, se crea la clase Excel haciendo uso de la librería open-source jxl. La clase Excel implementa el método estático *stats* el cual recibe como parámetro las diferentes estadísticas recogidas y las introduce en una hoja de cálculo creada junto con otros cálculos realizados a partir de las estadísticas.

La clase se ha implementado de tal forma que el fichero Excel se almacena lcoalmente en la carpeta "/Audit" y se abre automáticamente en el mismo proceso de ejecución.

A continuación, se observa en la Tabla 3-13 las conclusiones extraídas a partir de las estadísticas pasadas como parámetros. En ella se puede observar el tiempo medio de ejecución por noticia y método, el tiempo máximo de ejecución en una noticia y el número de total de menciones detectadas.

METODO	TIEMPO MEDIO	T MAX	TOTAL MENCIONES
LingPipe	20	1072	750
Apache	171	1140	259
Stanford	559	3082	328

Tabla 3-13 Conclusiones en Excel

	F	G	H	I	J	K	L	M
1	NOTICIAS		LingPipe		Apache		Stanford	
2	0		3	217	0	532	2	1416
3	1		1	140	1	353	0	850
4	2		1	136	0	363	6	912
5	3		0	8	0	240	0	831
6	4		1	12	0	236	1	1154
7	5		0	4	0	212	0	728
8	6		0	8	0	217	0	592
9	7		0	8	0	172	0	574
10	8		0	8	0	164	0	559
11	9		0	8	0	184	0	540
12	10		0	8	0	156	0	532

Tabla 3-14 Muestra de la tabla de estadísticas

En la Tabla 3-14 se puede apreciar la disposición de las estadísticas aportadas. En una disposición por columnas, se observa que la columna Noticias tiene referenciadas numéricamente cada noticia. Respecto a los tres métodos, cada uno cuenta con dos columnas, en la primera columna se hace referencia al número de menciones identificadas y en la segunda el tiempo de ejecución por noticia.

	A	B	C	D	E	F	G	H
1	METODO	TIEMPO MEDIO	T.MAX	TOTAL MENCIONES		NOTICIAS		LingPipe
2	LingPipe	20	1072	750		0		3
3	Apache	171	1140	259		1		1
4	Stanford	559	3082	328		2		1
5						3		0
6						4		1
7						5		0
8						6		0
9						7		0
10						8		0
11						9		0
12						10		0
13						11		1
14						12		0
15						13		1
16						14		1
17						15		0
18						16		0
19						17		3
20						18		3
21						19		0
22						20		0
23						21		1
24						22		0
25						23		0
26						24		0
27						25		246
28						26		0

Figura 3.16 Visión global de la hoja de cálculo Excel generada

3.4.11. Módulo Geolocalización

La geolocalización es el último paso a realizar en el ciclo de vida de la aplicación y a su vez una de las mayores motivaciones en la realización del sistema. No obstante, debido a que el lenguaje de programación con el que ha sido diseñado el software es Java, las fallas identificadas se posicionan geográficamente haciendo uso de la API de Google Static Maps.

Por tanto, la geolocalización se implementa mediante la creación de la clase GeoLocation en el paquete NER y se instancia como último paso en los casos de selección de opción LingPipe, Apache o Stanford dentro del menú switch de la clase Main. Su constructor exige que se le pase como parámetro la lista de fallas identificadas.

La clase GeoLocation dispone de una larga lista de atributos, de los cuales muchos se establecen en la instanciación de la clase, y otros parámetros como el centro de la imagen, el tamaño y la escala serán configurados automáticamente en función del número de fallas a identificar, gracias a la utilización de un bucle condicional creado y que decide en función de la distancia entre las coordenadas de las fallas estos parámetros.

A continuación, una vez instanciada la clase y establecidos los parámetros, para obtener la url con el formato que exige la API de Google Static Maps, se ejecuta el método *locate*, un método de tipo url y cuyo cometido consta precisamente en la construcción de la url y retorno de la misma.

```
base = base + "center="+ center +"&zoom=" + zoom + "&size=" + size + "&maptype=" + maptype +
        "&format=" + format + "&region=" + region + "&language=" + language + marcadores +
        "&sensor=" + sensor;
```

Figura 3.17 Ejemplo de construcción de la url

Para finalizar, la muestra del mapa se realiza mediante la implementación del método *picture*, el cual toma como parámetro la url devuelta y sustentándose en las propiedades de la interfaz gráfica de la clase Swing de Java, muestra en una imagen las fallas identificadas. Además, en caso de intentar geolocalizar una noticia donde no haya sido posible identificar ninguna mención a ningún monumento fallero, la imagen devolverá una imagen del centro de Valencia, con el Ayuntamiento marcado en color azul. En el caso afirmativo de disponer de una lista de fallas, éstas estarán dispuestas en el mapa con marcadores de color rojo.

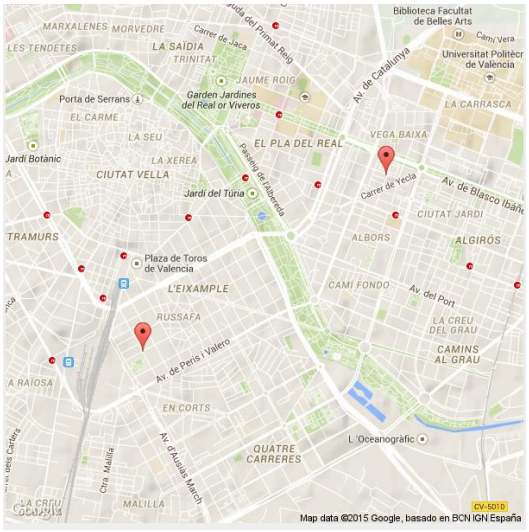

Figura 3.18 Geolocalización de 2 fallas

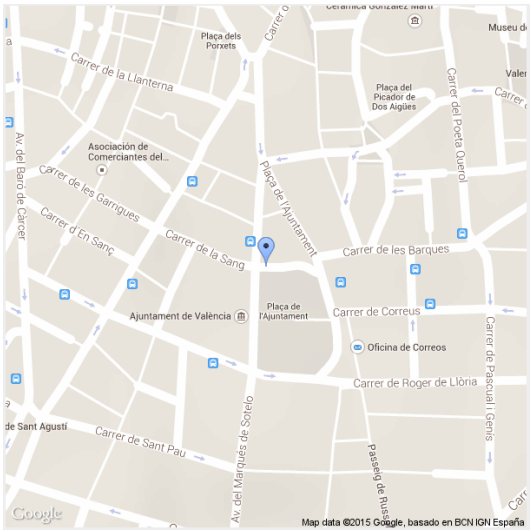

Figura 3.19 Geolocalización de lista vacía

3.4.12. Manejo de ficheros locales

Uno de las tomas de decisión cruciales en la forma, aspecto e implementación del software creado, fue la decisión de no crear una base de datos relacional y utilizar un servidor. Se trata de una decisión que deriva directamente en el manejo local de archivos y directorios. Este apartado pretende reunir el uso de los diferentes directorios, archivos y su disposición.

Dentro del directorio de trabajo, se han creado las carpetas “Archivos”, “Audit”, “Noticias”, “NoticiasPreparadas” y “Trainers”.

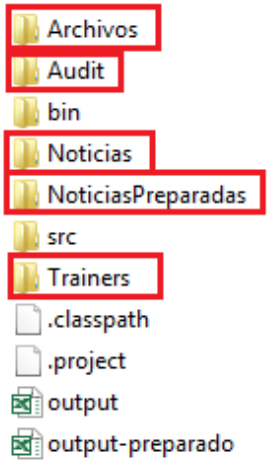

Figura 3.20 Directorio principal de trabajo

Dentro de "Archivos" se almacena el fichero "Anexo.txt", creado por el módulo Lector RSS; el fichero "Monumentos_falleros.JSON" y la carpeta Corpus, a su vez dividida en las carpetas Apache y Stanford donde se almacenan los corpus en sus respectivos formatos.

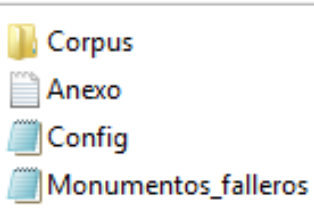

Figura 3.21 Contenido de la carpeta Archivos

"Audit" es un fichero de auditoría y comprobaciones. A lo largo de la elaboración del sistema se han realizado una gran cantidad de comprobaciones y debido al manejo de grandes volúmenes de información, era conveniente dirigir la salida hacia ficheros en lugar de a través de la consola.

"Noticias" es la carpeta que almacena todas las noticias extraídas de flujos de RSS y "NoticiasPreparadas" contiene diez noticias previas a la creación del lector de RSS y cuyo contenido no ha sido utilizado en la creación de modelos de clasificación NER.

Por último, la carpeta "Trainers" contiene a su vez dos carpetas "Apache" y "Stanford" que a su vez contienen los modelos de clasificación de sus respectivos métodos.

3.4.13. Librerías utilizadas

En programación, una librería (traducido erróneamente del inglés *library*) es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación (Java en este caso), que ofrece una interfaz bien definida para la funcionalidad que se invoca.

A diferencia de un programa ejecutable, el comportamiento que implementa una biblioteca no espera ser utilizada de forma autónoma (un programa sí: tiene un punto de entrada principal), sino que su fin es ser utilizada por otros programas, independientes y de forma simultánea. Por otra parte, el comportamiento de una biblioteca no tiene porqué diferenciarse en demasía del que pudiera especificarse en un programa. Es más, unas librerías pueden requerir de otras para funcionar, pues el comportamiento que definen

refina, o altera, el comportamiento de la biblioteca original; o bien la hace disponible para otra tecnología o lenguaje de programación.

Las bibliotecas pueden vincularse a un programa (o a otra librería) en distintos puntos del desarrollo o la ejecución, según el tipo de vínculo que se quiera establecer.

En este proyecto se ha hecho uso de las siguientes librerías:

- **gson-2.3.1.jar:** Librería empleada para la lectura de ficheros de formato JSON.
- **jsoup-1.8.1.jar:** Librería empleada para depurar las noticias de código XML al ser extraídas de los feeds RSS.
- **jxl.jar:** Librería de funcionamiento similar a "Java Swing" continente de funciones relativas a generación y manipulación de hojas de cálculo Microsoft Excel.
- **lingpipe-4.1.0.jar:** Librería de desarrollo de NER de LingPipe.
- **opennlp-tools-1.5.3.jar:** Librería de desarrollo de NER de Apache Open NLP.
- **stanford-ner-3.5.0.jar:** Librería de desarrollo de NER de Stanford NLP.

Capítulo 4. Estudio comparativo

En este capítulo se aborda en primer lugar la evolución padecida por cada una de las herramientas NER en búsqueda de su módelo de clasificación óptimo y, por último, se realiza un análisis comparativo basado en las estadísticas obtenidas entre las tres herramientas NER desarrolladas.

4.1. Desarrollo evolutivo de los modelos

En este apartado se detalla la evolución sufrida por los tres métodos de reconocimiento de nombres de entidades en la búsqueda de su modelo ideal.

4.1.1. LingPipe

La herramienta NER de LingPipe es de entre las tres, aquella cuyos cambios sufridos repercuten de forma más inmediata y directa al no depender de un modelo.

Una ventaja implícita en los servicios y herramientas de Reconocimiento de Nombres de Entidades es, precisamente, el evitar la tediosa laboriosidad que requiere crear un conjunto de reglas ‘a mano’. Sin embargo, el uso de LingPipe se perfecciona y precisa de esta forma al no estar basado en el entrenamiento de un modelo de clasificación como así son los métodos de Apache y Stanford.

Por esta razón, en el desarrollo de la herramienta LingPipe se han llevado a cabo una gran variedad de pruebas en búsqueda del modelo óptimo. Ya se describen más detalladamente algunas de las fórmulas acometidas en el punto 3.4.7, en el caso de las

entradas ‘extraordinarias’[6] en el diccionario en los casos de menciones bilingües, menciones por uso de sobrenombre y entradas dedicadas en función de su cuota de menciones en noticias.

Se puede observar en la siguiente tabla (Tabla 4-1 Comparación de uso de entradas especiales en el diccionario) el incremento de la precisión. A cambio de un incremento en el tiempo de ejecución por noticia de 1ms, obtenemos un 7,3% más de menciones. Una mejora en la precisión considerable si además tenemos en cuenta que las entradas ‘extraordinarias' en el diccionario se han realizado únicamente para un 64,56%.

LINGPIPE	No Esp	Si Esp	Incremento
Menciones identificadas	699	750	7,3%
Tiempo de ejecución medio (ms)	18	19	5,6%

Tabla 4-1 Comparación de uso de entradas especiales en el diccionario

Siguiendo la misma progresión, en caso de realizar entradas ‘extraordinarias’ para los 350 monumentos falleros y despreciando el tiempo de elaboración y extensión del código, se estima que se detectarían 79 menciones más respecto a la no utilización de diccionario. Es decir, con 350 entradas dedicadas, se cubriría aproximadamente un hipotético 11,3% más de menciones.

4.1.2. Apache Open NLP

En el caso de la herramienta NER basada en la API proporcionada por Apache Open NLP, la única mejora posible de cara a aumentar el número de menciones reconocidas y la velocidad de ejecución es a través de modificaciones en el corpus. Para ello, se han creado tres modelos de clasificación NER distintos.

En la creación del corpus de Apache, se recomienda que además del formato especificado, la longitud del corpus sea de al menos de 15.000 líneas. Sin embargo, al ser las Fallas de Valencia una temática muy concreta y el periodo de generación de noticias relacionadas muy breve, no se ha podido alcanzar dicha cifra. Desde el equipo de ProDevelop se han facilitado las noticias captadas y almacenadas en su base de datos, con el objetivo de intentar aumentar las líneas de los corpuses. No obstante, de las 975 noticias

[6] La relación de las entradas ‘extraordinarias’ se encuentra detallada en el Anexo A.

facilitadas, únicamente 23 no se encuentran entre las 1.068 captadas por el software desarrollado en este proyecto.

El primero de los modelos creados (*es-fallas-Red.bin*) tiene su origen en un error de compilación en la creación del módulo lector RSS. Una serie de noticias entraron repetidamente en el fichero "Anexo.txt" utilizado más tarde para realizar el corpus de Apache. El error fue subsanado sin graves consecuencias y, además, nos proporcionaba una mayor extensión del corpus, si bien su información era redundante. Este error, derivó en la idea de aprovechar esa extensión y entrenar el corpus con información redundante. Una vez creado el corpus, se creó un programa al margen del software del proyecto con el fin de cambiar las menciones envueltas en 'tags' de forma iterativa utilizando las 350 fallas registradas. Por tanto, en este proyecto a modo de prueba se utiliza un modelo con información redudante, con la excepción de haber alterado las menciones dentro de las líneas de texto redundantes.

El segundo modelo creado (*es-fallas-Extended.bin*) se trata de un corpus producido por la suma del corpus creado en primer lugar sin líneas redundantes y al que se le añaden las líneas correspondientes a las 23 noticias captadas por la base de datos de LiveFallas.

En último lugar, basándonos en la célebre frase de Albert Einstein "Si buscas resultados distintos, no hagas siempre lo mismo." (Albert Einstein, físico de origen alemán y Premio Nobel de Física en 1921) se decide crear un corpus a partir de las 975 facilitadas por LiveFallas y las 1.068 halladas por el módulo Lector RSS. Esta idea nace del hecho de que una parte de las noticias generadas en Fallas son creadas en primera instancia por agencias de prensa, las cuales después distribuyen a los medios de comunicación escritos, quienes realizan pequeñas modificaciones en las noticias. Por ello, se "monta" un corpus analizando línea a línea primero los dos corpus creados anteriormente y a continuación comparando con las noticias facilitadas por LiveFallas. De esta forma se obtiene el tercer modelo empleado (es-fallas-Extended-Clean.bin).

Una vez creados los tres modelos se han hecho pruebas de ejecución y clasificación NER con ellos. Los resultados, son los que aparecen en la tabla mostrada a continuación:

APACHE	Extended	Extended-Clean	Red
Menciones identificadas	102	106	104
Tiempo de ejecución medio (ms)	179	197	208
Tamaño (kB)	823	833	942
Extensión del corpus en líneas	13.943	14.385	16.216

Tabla 4-2 Comparación de los tres modelos de Apache

La actuación de los tres modelos es similar, si bien es cierto que el modelo Extended-Clean es el que más menciones es capaz de detectar. Respecto a los tiempos de ejecución, hay una diferencia despreciable de 18ms por noticia entre el menor tiempo y el correspondiente a Extended-Clean.

4.1.3. Stanford NLP

En lo relativo a la determinación del modelo de clasificación óptimo basado en Stanford NLP, se ha llevado a cabo un procedimiento similar al ejecutado para el caso de Apache.

Haciendo uso del módulo NER Logistics, a partir de los tres corpuses creados en Apache se han creado corpuses para Stanford y, posteriormente generado tres modelos de clasificación para Stanford NER:

- fallas-Extended.ser.gz
- fallas-Extended-Clean.ser.gz
- fallas-Red.ser.gz

La procedencia de los corpuses y su elaboración ya han sido explicados en el punto anterior 4.1.2 por lo que en este apartado se procederá a comentar la evaluación de la clasificación NER mediante sendos métodos.

En primer lugar, se puede observar en la Tabla 4-3 que el tamaño de los ficheros de clasificación, al contrario que en Apache, no son proporcionales a la extensión del corpus.

Stanford	Extended	Extended-Clean	Red
Menciones identificadas	62	62	153
Tiempo de ejecución medio (ms)	594	600	564
Tamaño (kB)	12.581	12.441	12.416

Tabla 4-3 Comparación de los tres modelos de Stanford

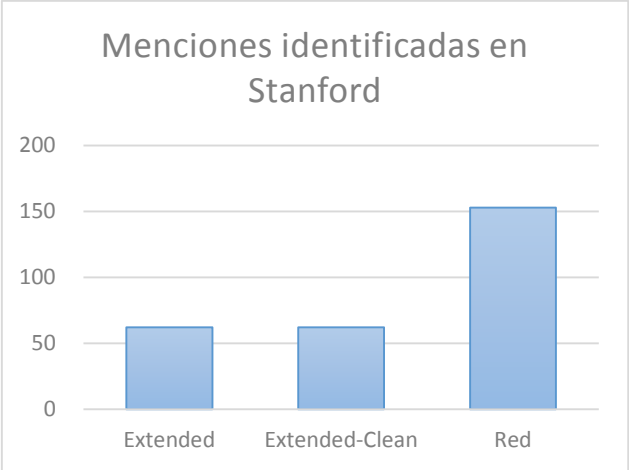

Figura 4.1 Gráfico de comparación de menciones

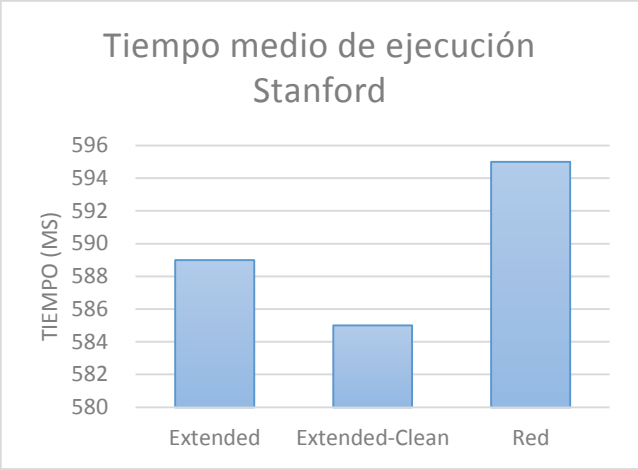

Figura 4.2 Gráfico de comparación de tiempos de ejecución

La comparativa gráfica de los tres modelos refleja la superioridad del modelo basado en información redundante respecto a los otros modelos. El modelo Red no sólo mejora notablemente la actuación en términos de clasificación NER, siendo un 114% superior a los otros dos modelos. Igualmente, respecto a los valores de tiempos de ejecución medio, existe una variación de 10ms entre los modelos Extended-Clean y Red – un 1.17% superior respecto al tiempo medio de ejecución Extended-Clean – y considerada por tanto, despreciable.

Otro valor destacable respecto de la herramienta Stanford es la fluctuación de las medidas de tiempos de ejecución. Esta oscilación se debe a que Stanford se basa en procesos heurísticos.

Tiempos de ejecución de Stanford	1	2	3	4
Extended	594	586	584	592
Extended-Clean	600	576	591	572
Red	564	614	583	621

Tabla 4-4 Medición de tiempos de ejecución Stanford

Tiempos de ejecución de Stanford	MEDIA	Oscilación Max
Extended	**589**	**10**
Extended-Clean	**585**	**28**
Red	**595**	**57**

Tabla 4-5 Estadísticas de tiempos de ejecución Stanford

En la Tabla 4-4 se observa la fluctuación respecto al tiempo de ejecución para cuatro ejecuciones con cada modelo Stanford. Finalmente, en la Tabla 4-5 observamos la oscilación máxima producida y el tiempo de ejecución medio para cada modelo.

Una vez analizadas las gráficas y la actuación de los tres modelos creados para analizar noticias con Stanford, se determina que el modelo Red es el que mejor cumple con las características.

4.2. Análisis comparativo a tres

En este apartado se realiza un estudio comparativo a tres entre la herramienta LingPipe y las herramientas Apache y Stanford utilizando sendos modelos óptimos determinados en el punto anterior 4.1.

Para establecer un criterio comparativo unificado, se ha realizado un estudio estadístico siguiendo las recomendaciones aportadas por la comunidad de desarrolladores www.stackoverflow.com donde se recomienda no utilizar modelos de clasificación sobre textos incorporados al propio corpus. Con este motivo, se han preservado 15 noticias al margen de la creación de los distintos corpuses y realizar un estudio comparativo lo más fidedigno posible.

Junto a la herramienta LingPipe, en el estudio a tres se han utilizado para Apache Open NLP y Stanford NLP los modelos *Extended-Clean* y *Red* respectivamente.

Los resultados globales del estudio son los mostrados a continuación:

MODELO	TIEMPO MEDIO	T MAX	TOTAL MENCIONES
LingPipe	171	469	155
Apache Open NLP	259	469	45
Stanford NLP	946	1.538	93

Tabla 4-6 Comparación de ejecución por métodos

En la Tabla 4-6 se observa la mayor velocidad de ejecución de LingPipe y de igual forma, la mayor efectividad en reconocimiento de nombres de monumentos falleros. Apache, sin embargo, presenta unos tiempos de ejecución moderados y una pobre eficiencia en cuanto a la detección de menciones, ofreciendo tanto Stanford como LingPipe resultados muy superiores.

En un desglose de la ejecución por noticia, la actuación de los tres métodos es la siguiente:

Menciones por Noticia	Menciones reales	LingPipe	Apache	Stanford
Noticia 1	25	24	6	6
Noticia 2	14	11	5	8
Noticia 3	40	31	5	9
Noticia 4	4	5	1	0
Noticia 5	1	1	0	0
Noticia 6	3	3	3	0
Noticia 7	12	8	0	5
Noticia 8	37	33	8	22
Noticia 9	10	9	5	15
Noticia 10	10	9	6	8
Noticia 11	1	1	0	0
Noticia 12	15	11	5	8
Noticia 13	1	1	0	0
Noticia 14	8	6	0	1
Noticia 15	1	2	1	11

Tabla 4-7 Análisis desglosado por noticias

Al ser LingPipe la herramienta NER más efectiva a priori de las tres, se procede a analizar ésta en detalle. Tal y como se aprecia en la Tabla 4-7, a pesar de resultar ser una herramienta notablemente eficiente, se intuyen falsos positivos.

Los falsos positivos tienen lugar en debido a nombres determinados de algunas fallas, cuyos nombres son breves y de uso común, dando lugar a error. Un caso común, por ejemplo, ocurre con Plaza de la Merced. Dentro de la implementación del módulo NER Set-Up, en su clase Normalizar se eliminan las palabras clave Plaza y Avenida. Estas dos palabras son de corte demasiado general y sus acepciones en modo de abreviación extensas. En el caso de la falla "Plaza de la Merced", queda únicamente la palabra "Merced" como entrada al diccionario. Posteriormente, las transformaciones realizadas por LingPipe, en ocasión dan lugar a error.

Este tipo de fallos, no obstante, debe asumirse en el Reconocimiento de Nombres de monumentos falleros, ya que por lo general muchas fallas reciben su nombre por el lugar exacto de las calles donde éstas se plantan. Las calles a su vez, toman en un gran

número de casos, de nombres y lugares geográficos. Pudiendo aparecer, menciones no relacionadas con los monumentos falleros, dando lugar a falsos positivos.

Por consiguiente, se ha analizado la eficiencia de la herramienta NER LingPipe por noticias, haciendo especial hincapié en la detección de las palabras no detectadas y en los falsos positivos, confirmando que las menciones no detectadas, por norma general, pertenecen a menciones de monumentos falleros cuyos nombres han sido reducidos a la mínima expresión. En el caso de modificar el algoritmo de la herramienta LingPipe para detectar estas mínimas menciones, induciríamos el incremento de falsos positivos.

La Tabla 4-8 refleja precisamente la eficiencia de la herramienta LingPipe y la aparición de falsos positivos.

LingPipe	Menciones reales	Identificadas	Falsos Positivos	Eficiencia
Noticia 1	25	24	1	92%
Noticia 2	14	11	0	79%
Noticia 3	40	31	0	78%
Noticia 4	4	5	1	100%
Noticia 5	1	1	0	100%
Noticia 6	3	3	0	100%
Noticia 7	12	8	0	67%
Noticia 8	37	33	0	89%
Noticia 9	10	9	0	90%
Noticia 10	10	9	0	90%
Noticia 11	1	1	0	100%
Noticia 12	15	11	0	73%
Noticia 13	1	1	0	100%
Noticia 14	8	6	0	75%
Noticia 15	1	2	1	100%
				88,06%

Tabla 4-8 Eficiencia de LingPipe por noticia

Con el objetivo de refutar los hechos expuestos, los tres falsos positivos, se dan con los siguientes monumentos falleros:

- Ribera – Convento Santa Clara "Telefónica"
- Exposición – Micer Mascó
- Plaza del Pilar

Capítulo 5. Otras aplicaciones

En el presente apartado se comenta el (posible) uso de servicios de geolocalización de nombres de entidades reconocidas en noticias en otros ámbitos profesionales.

5.1. Agencias de comunicación, prensa y R.R.P.P.

A día de hoy, todo tipo de empresa, grande o pequeña, dedica parte de sus recursos a la publicidad. Directa, mediante agencias de publicidad, o indirecta, mediante agencias de comunicación y relaciones públicas.

El cometido final de una agencia de comunicación y relaciones públicas consiste en brindar un soporte estratégico para sus clientes, es decir, las marcas contratantes y establecer vínculos y relaciones positivas y de proyección con el usuario final y, además, respaldar las estrategias de marketing de empresas contratantes, encargarse de la introducción de productos, reposicionamientos, organización de eventos y conferencias de prensa, entre otras actividades.

Es decir, realizan una gestión profesional de la comunicación para alcanzar los objetivos planteados, al aportar una mirada externa a la empresa y soluciones sin prejuicios a las diversas necesidades internas y externas de la organización. [28]

La utilización de una herramienta de geolocalización de menciones halladas en flujos de noticias les aportaría información para realizar un estudio de mercado e influencia en clave geográfica. Es decir, valerse de esta herramienta les facilitaría la distribución de esfuerzos y recursos y emplear éstos en aquellas áreas geográficas donde no exista una influencia de las marcas contratantes.

5.2. Tráfico

El estado de las carreteras es un gran foco de atención a diario tanto para el sector del transporte, como para el ciudadano corriente en sus desplazamientos al trabajo o también de vacaciones. La variedad de motivos en realidad por la que mantenerse informado del estado de las carreteras es muy amplia.

La evolución de la tecnología nos permite ya que actualmente tengamos la posibilidad de consultar el estado de las carreteras en tiempo real, bien desde un dispositivo móvil o bien a través de la radio si nos encontramos en un transporte. No obstante, estamos siendo testigos de un crecimiento incipiente del tan llamado "Internet de las cosas". En un futuro próximo, los vehículos irán provistos de conexión a Internet y la información de navegación será susceptible de ser actualizada en tiempo real.

Es precisamente esta tesitura un marco ideal de aplicación de la herramienta creada en este proyecto. En un vehículo con conexión a internet y pantalla de navegación, podría implementarse un sistema en el que leyendo información relativa al estado del tráfico y las carreteras, fuese capaz de geolocalizar menciones de carreteras y kilómetro concreto donde ocurre un suceso determinado. Una vez concretada la posición, el sistema volcaría la información a la pantalla de navegación y tomar decisiones.

5.3. Desastres naturales

Los fenómenos naturales como las lluvias torrenciales, incendios forestales, movimientos sísmicos o tsunamis, copan la atención mediática internacional, siendo objeto de atención desde el mismo instante en el que ocurren.

Debido a las consecuencias devastadoras generalmente producidas por los desastres naturales, asolando todo a su paso y provocando cuantiosas pérdidas materiales, agujeros económicos cuantiosos e, incluso, lamentablemente desapariciones y fallecimientos de seres humanos, la atención no sólo por parte de los medios sino por los propios consumidores de estos, es notable.

Por ello, con el objetivo de mejorar la calidad de la información relativa a desastres naturales, sería interesante implementar un sistema de geolocalización a partir de las menciones de localizaciones geográficas identificadas en los cuerpos de las noticias, aportando una perspectiva espacial de los lugares donde estos ocurren.

5.4. Seguridad Estatal

Internet y las redes sociales han transformado el mundo tal y como lo conocíamos para siempre. Hoy en día, cualquier persona, anónima o no, puede valerse de Internet como altavoz para expandir el mensaje que desea transmitir.

Precisamente este anonimato, unido al fácil acceso a la red y a la disponibilidad de poder emitir información desde cualquier lugar lo que ha terminado en hacer de Internet un escaparate para todo tipo de conductas delictivas de bajo calado ético, ya sea apología del terrorismo, incitación a la anorexia o bulimia o pornografía infantil entre otros.

Es por esta razón, por la que los cuerpos de seguridad estatales a nivel internacional crearon unidades especializadas de delitos telemáticos. Estas unidades disponen de herramientas y técnicas de inspección para monitorizar la actividad en la red e investigar las actividades delictivas acaecidas en el mundo virtual.

Dentro de este abanico de herramientas y técnicas encajaría una adaptación de la aplicación creada en este proyecto. La monitorización de noticias publicadas mediante *feeds* RSS de interés en busca de palabras clave, creando a su vez un nexo geográfico gracias al rastreo IP para su posterior geolocalización ofrecería una visión espacial de aquellos lugares donde se cometen los delitos. De esta forma, apoyándose en claves geográficas, podrían buscarse patrones de conducta o localizar directamente actividades sospechosas.

Capítulo 6. Conclusión

En este capítulo, como colofón del proyecto, se expone en primer lugar una recopilación de los resultados obtenidos, a continuación se presentan los cambios necesarios para poder integrar la herramienta en un entorno empresarial y, para finalizar, una conclusión final a modo de valoración del proyecto.

6.1. Recapitulación de resultados

Uno de los objetivos principales del presente proyecto consistía en hallar entre LingPipe, Apache Open NLP y Stanford NLP, la herramienta más adecuada de cara a encontrar el mayor número de menciones posibles en canales RSS.

En primer lugar, la herramienta NER basada en LingPipe es la que mejores resultados ha demostrado, tanto en términos de tiempo medio de ejecución, como en capacidad para identificar menciones de monumentos falleros en noticias.

En segundo lugar, la herramienta mixta creada a partir de Apache Open NLP y envuelta en LingPipe, es la que muestra resultados más insatisfactorios. El tiempo medio de ejecución es considerablemente elevado respecto a LingPipe y su capacidad de reconocimiento, nimia.

Por último, la herramienta creada a partir de Stanford NLP, a pesar de no cumplir los requisitos de velocidad de ejecución deseados, sí es un método apto en términos de identificación de monumentos falleros aunque sin alcanzar la eficiencia demostrada por LingPipe.

Por tanto, basado en el análisis realizado en el proyecto, se estima que para el caso concreto de reconocimiento de menciones de monumentos falleros en noticias, LingPipe es la herramienta más adecuada y eficiente, aportando ésta resultados ostensiblemente superiores a los otros dos métodos desarrollados. Sin embargo, al no haberse podido elaborar un corpus para Apache cumpliendo su longitud mínima recomendada, queda abierta la puerta de que, a largo plazo, mejore su eficiencia, al igual que con un corpus más extenso Stanford también podría llegar a aumentar su precisión.

6.2. Integración en entorno empresarial y trabajo futuro

Puesto que el presente proyecto ha sido diseñado y realizado bajo un contexto académico, es necesario destacar que existen aspectos con un amplio margen de mejora en el caso de querer aplicar el sistema creado en un entorno empresarial.

En el punto 3.3.3 de este documento se exponen los criterios de diseño y decisiones clave adoptadas en la concepción de la aplicación. Precisamente, una de las necesidades de este proyecto en la integración de la herramienta en un entorno empresarial es la de abandonar el manejo de ficheros almacenados localmente. Como contrapartida se basaría el manejo de datos de la aplicación mediante bases de datos y un servidor, aportando al sistema mayor solidez y autosuficiencia.

Otra mejora podría ser la adaptación de la herramienta a Android e iOS y entrar en el mercado de las aplicaciones para móviles. Además, basando la aplicación en Android e iOS dispondríamos de la posibilidad de disponer de la API de Google Maps y su carga dinámica y no restringir la geolocalización al uso de los mapas estáticos.

6.3. Conclusión final

Se puede determinar que los objetivos del proyecto se han cumplido de forma notable, habiéndose creado una aplicación capaz de reconocer y geolocalizar correctamente el 88.06% de las menciones reales halladas en noticias.

Igualmente, cabe destacar la falta de escalabilidad de la herramienta más eficiente en el ámbito del reconocimiento de menciones de fallas, al ser LingPipe una herramienta basada en diccionario y pudiendo suponer un problema en el caso de querer cubrir un grupo más numeroso en una atmósfera distinta.

Apéndice A. Fallas de trato especial

En este apéndice se detalla por categorías la relación de aquellas fallas que han recibido un trato especial en el diccionario de la clase Matcher, en la implementación de la herramienta de clasificación NER de LingPipe.

- **Sección Especial**

- Plaza del Pilar
- Convento Jerusalén – Matemático Marzal
- Cuba – Literato Azorín
- Exposición – Micer Mascó
- Almirante Cadarso – Conde Altea
- Regne de Valencia – Duque de Calabria
- Sueca – Literato Azorín
- Na Jordana
- Malvarrosa – Antonio Ponz Cavite
- Monestir de Poblet – Aparicio Albiñana
- Jorge Comín – Serra Calderona
- Archiduque Carlos - Chiva

- **Sección 1A**

- Barraca – Espadán
- Duque de Gaeta – Pobla de Farnals
- Plaza del Mercado Central
- Justo Vilar – Plaza Mercado Cabanyal
- San Vicente – Periodista Azzati
- Plaza Mercado Monteolivete
- Maestro Gozalbo – Conde Altea
- Císcar – Burriana
- Grabador Esteve – Cirilo Amorós
- Quart Extramuros – Velázquez
- Ceramista Ros – J.M. Montes Lerma
- Plaza Obispo Amigó – Cuenca
- S. Rusiñol – Conde Lumiares
- Ramiro de Maeztu – Leones
- Aras de Alpuente – Castell de Pop
- Quart – Palomar (Plaza Santa Palomar)
- Federico Mistral – Murta
- Ribera – Convento Santa Clara

▪ Sección 1B

- Pie La Cruz – Don Juan Vilarrasa
- Plaza Pintor Segrelles
- Oltá – Juan Ramón Jimenez
- Barrio de San Isidro
- Joaquín Costa – Conde Altea
- Islas Canarias – Lo Rat Penat
- Doctor Molinell – Alboraya
- Santa María Micaela – Martín el Humano
- Plaza de la Reina – Paz – San Vicente
- Isabel la Católica – Cirilo Amorós
- San José Montaña – Teruel
- Santa Genoveva Torres – Arquitecto Tolsa – Alfahuir
- Pintor Stolz – Burgos
- Alquerías de Bellver – Garbí
- Pintor P. Capuz – Fontanares
- Arzobispo Olaechea – S. Marcelino
- Pizarro – Cirilo Amorós
- Prolongación Alameda – Avenida Francia
- Marqués de Montortal – José Esteve

▪ Bilingüismo

- Quart Extramuros - Velazquez
- Plaza del Negrito
- Arzobispo Olaechea – S. Marcelino
- Duque de Gaeta – Puebla de Farnals

▪ Sobrenombres

- Monestir de Poblet – Aparicio Albiñana → "L'antiga"
- Falla Jorge Comín – Serra Calderona → "Nou Campanar"
- Ribera – Convento Santa Clara → "Telefónica"
- Ángel Guimerá – Vila Prades → "Arrancapins"

Apéndice B. Lista de figuras

Apéndice C. Lista de tablas

Bibliografía

[1] http://blogs.salford.ac.uk/business-school/geolocation-api-business-benefit/

[2] http://langtech.jrc.ec.europa.eu/Documents/0408_Kimler_Thesis-GeoCoding.pdf

[3] http://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)#Filosof.C3.ADaAs

[4] http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/Caracter%C3%ADsticas_del_lenguajeAs

[5] https://geekytheory.com/json-i-que-es-y-para-que-sirve-json

[6] http://json.org/json-es.htmlSas

[7] http://es.wikipedia.org/wiki/Extensible_Markup_Language

[8] http://www.hipertexto.info/documentos/xml.htm

[9] http://en.wikipedia.org/wiki/RSS

[10] Holzner, Steven. Secretos de RSS. Anaya Multimedia, 2006. ISBN 9788441521308

[11] http://en.wikipedia.org/wiki/Natural_language_processing

[12] http://alias-i.com/lingpipe/

[13] https://opennlp.apache.org/documentation/1.5.3/manual/opennlp.html#tools.namefind

[14] http://www.programcreek.com/2012/05/opennlp-tutorial/

[15] http://nlp.stanford.edu/software/CRF-NER.shtml#Download

[16] https://blogs.nd.edu/wilkens-group/2013/10/15/training-the-stanford-ner-classifier-to-study-nineteenth-century-american-fiction/

[17] http://www.googlemaps.es/?page_id=3

[18] http://es.wikipedia.org/wiki/Eclipse_(software)

[19] http://www.queesexcel.net/

[20] http://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software

[21] http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software

[22] http://es.wikipedia.org/wiki/Desarrollo_en_cascada#Variantes

[23] http://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente

[24] http://es.wikipedia.org/wiki/Modelo_de_prototipos

[25] http://html.rincondelvago.com/desarrollo-orientado-a-prototipos.html

[26] http://es.wikipedia.org/wiki/Desarrollo_en_espiral

[27] http://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema

[28] http://www.altonivel.com.mx/23760-que-hace-una-agencia-de-relaciones-publicas.html

[29] https://developers.google.com/maps/documentation/staticmaps/?csw=1#quick_example