RECUBRIMIENTO Y NORMALIZACIÓN DE RECURSOS LINGÜÍSTICOS:

APLICACIÓN A LAS ANOTACIONES MORFOSINTÁNTICAS Y SINTÁCTICAS DE FREELING

BALLESTEROS CALVO, ALICIA

CÁRCAMO ESCORZA, GUILLERMO

DUOBERT COLLAZOS, EMILIO

FACULTAD DE INFORMÁTICA, UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO FIN DE GRADO EN INGENIERÍA DEL SOFTWARE

SEPTIEMBRE 2013

Director: Antonio Pareja Lora

1. RESUMEN

Existen muchos conflictos y problemas que impiden que las herramientas y las anotaciones lingüísticas puedan interoperar y que, por lo general, provienen del enorme número de factores que se tienen en cuenta en su diseño y su desarrollo. En general, en el mundo de las herramientas lingüísticas se echan en falta servicios o APIs fácilmente manejables para el procesamiento automático del lenguaje, así como herramientas que proporcionen anotaciones normalizadas. Todo esto, dificulta el análisis y la comprensión de los datos obtenidos, así como su interoperabilidad. Además, el desarrollo de nuevas herramientas lingüísticas tiene un coste muy elevado.

En este trabajo proponemos, para resolver estos problemas, envolver un software ya existente, accesible y gratuito, con una nueva capa que lo dote de interoperabilidad y que proporcione anotaciones normalizadas. Además, para obtener una envoltura independiente del lenguaje de programación y de la plataforma tecnológica donde sea utilizada, se ha implementado dicha envoltura en forma de servicio web. El presente proyecto, *"Recubrimiento y normalización de recursos lingüísticos: aplicación a las anotaciones morfosintácticas y sintácticas de FreeLing*", tiene por objetivo incluir una capa de abstracción y normalización a las funciones de anotación morfosintáctica y sintáctica para el español de **FreeLing**, una librería de procesamiento automático multilingüe.

Palabras Clave: FreeLing, servicio web, MAF, SynAF, interoperabilidad, normalización.

2. ABSTRACT

There are many conflicts and problems that prevent linguistic tools and annotations from interoperating. Mostly, these conflicts and problems come from the number of factors that are taken into account in their development and design. The area of linguistic tool is missing some services or APIs that are easily manageable, as well as some tools that provide standardized annotations. All this makes it difficult to analyze and understand the data obtained and the interoperation of both tools and annotations. In addition, developing new linguistic tools has a high cost.

In order to overcome all these problems, in this work, we propose to wrap some freely available software with a new layer that provides standardized annotations and makes the tool more interoperable. Besides, we propose to implement this wrapping layer as a web service, so that the resulting wrapper is language and platform independent.

The present B. Sch. thesis, entitled "Wrapping *and standardizing linguistic resources: application to the morphosyntactic and syntactic annotations of FreeLing*", aims at including a layer of abstraction and standardization to the morphosyntactic and syntactic annotation functionalities for Spanish of FreeLing, which is a library for the automatic processing of several languages.

Key Words: FreeLing, web services, MAF, SynAF, interoperability, standardization.

3. INTRODUCCIÓN

La gramática es el estudio de las reglas y principios que determinan el uso de las lenguas y la organización de las palabras dentro de una oración[1]. También se denomina así al conjunto de reglas y principios que gobiernan el uso de un lenguaje determinado. Así, cada lengua tiene su propia gramática.

La gramática es una parte del estudio general del lenguaje denominado lingüística. Tradicionalmente, el estudio de la lengua se divide en cuatro niveles: [2]

- **Nivel fonético:** se encarga del estudio de la naturaleza acústica y fisiológica de los sonidos.
- **Nivel morfológico:** estudia (1) la estructura interna de las palabras para delimitar, definir y clasificar sus unidades; (2) las clases de palabras a las que dan lugar; y (3) los procesos y patrones de formación de nuevas palabras.
- **Nivel sintáctico:** estudia cómo se combinan las palabras.
- **Nivel semántico:** estudia la codificación del significado dentro de las expresiones lingüísticas
- **Nivel discursivo y pragmático:** estudiado por la filosofía del lenguaje, la filosofía de la comunicación y la psicolingüística o psicología del lenguaje, se interesa por el modo en el que el contexto influye en la interpretación del significado de las expresiones lingüísticas.

Por su parte, el procesamiento del lenguaje natural es un campo amplio y complejo, que cada vez cobra mayor importancia en nuestro mundo, y que intenta analizar y comprender el lenguaje de forma automática, especialmente en la red.

---

[1] Wikipedia: Gramática. https://es.wikipedia.org/wiki/Gramática. Consultado en febrero de 2013.

[2] Wikipedia: Lingüística. https://es.wikipedia.org/wiki/Lingüistica. Consultado en febrero 2013.

Actualmente existen diversas herramientas lingüísticas encargadas de analizar el lenguaje y ofrecer el resultado del mismo a través de un cierto formato, un etiquetado o lo que es lo mismo, una anotación concreta que representa la salida del análisis realizado.

Dichas herramientas analizan distintos niveles de la lengua, pero la realidad es que se trata de desarrollos independientes, sin pautas comunes, que no permiten que las aplicaciones puedan interactuar y comunicarse.

De los niveles descritos, el análisis gramatical cubre el nivel sintáctico y morfosintáctico (véase el apartado 4.4.2.1), en los cuales nos centraremos en este trabajo.

En general, las herramientas lingüísticas presentan las siguientes carencias:

- **Falta de servicios o APIs fácilmente manejables** para el procesamiento automático del lenguaje, lo que complica la personalización del software.
- **Falta de herramientas que proporcionen anotaciones normalizadas.** Esto dificulta el análisis y la comprensión de los datos obtenidos, así como su interoperabilidad. Además el desarrollo de nuevas herramientas para el campo de la gramática tiene un coste muy elevado.

Las carencias expuestas hacen patente la necesidad de incluir ciertas características en las herramientas lingüísticas, imprescindibles para proporcionar vías de comunicación entre ellas. Estas características indispensables son la interoperabilidad y la normalización, que deben ser los principios claves para el desarrollo de las futuras herramientas, por lo que se han considerado esenciales en este proyecto.

Por un lado, “la interoperabilidad es la habilidad de organizaciones o sistemas dispares y diversos para interaccionar con objetivos consensuados, comunes y con la finalidad de obtener beneficios mutuos. La interacción implica que las organizaciones

involucradas compartan información y conocimiento a través de sus procesos de negocio, mediante el intercambio de datos entre sus respectivos sistemas de tecnología de la información y las comunicaciones" [1].

Por otro lado, la normalización (*standarization*), según la ISO (*International Organization for Standarization*), es la actividad que tiene por objeto establecer, ante problemas reales o potenciales, disposiciones destinadas a usos comunes y repetidos, con el fin de obtener un nivel de ordenamiento óptimo en un contexto dado, que puede ser tecnológico, político o económico[3].

La normalización persigue fundamentalmente tres objetivos:

- **Simplificación:** para reducir el número de modelos y quedarse únicamente con los más necesarios.
- **Unificación:** para permitir su intercambio a nivel internacional.
- **Especificación:** para evitar errores de identificación, creando un lenguaje claro y preciso.

Las elevadas sumas de dinero que los países desarrollados invierten en los organismos normalizadores, tanto nacionales como internacionales, son una prueba de la importancia que tiene la normalización[4].

Ahora bien, la interoperabilidad y la normalización son instrumentos que facilitan el acceso a servicios y utilidades enmarcados en un área de trabajo específico. En este caso, el marco de desarrollo es el área de la gramática.

El presente proyecto, *"Recubrimiento y normalización de recursos lingüísticos: aplicación a las anotaciones morfosintácticas y sintácticas de FreeLing*", tiene por

[3] Wikipedia: Normalización. http://es.wikipedia.org/wiki/Normalizaci%C3%B3n. Consultado en febrero de 2013.

[4] Véase la nota al pie anterior.

objetivo incluir una capa de abstracción a la funcionalidad sintáctica y morfosintáctica, que nos ofrece la librería de desarrollo de procesamiento multilingüe automático **FreeLing**.

Se pretende que una serie de servicios ya desarrollados cubran las principales funciones de análisis y anotación gramatical, más concretamente las funciones de análisis sintáctico y morfosintáctico para el español. También se pretende tratarlos de tal manera que ofrezcan a cualquier tipo de usuarios la posibilidad de utilizarlos. Además, el sistema resultante deberá proporcionar resultados normalizados, que faciliten su interoperación con otras anotaciones y aplicaciones.

Teniendo en cuenta lo anterior proponemos envolver un software ya existente, accesible y gratuito (la librería FreeLing), con una nueva capa que lo dote de interoperabilidad y que proporcione anotaciones normalizadas. Además, se busca obtener una envoltura independiente del lenguaje de programación y de la plataforma tecnológica donde sea utilizada, y que permita a otros usuarios mejorar los servicios y funciones disponibles actualmente en esta área de trabajo. Se pretende ser un impulso de normalización e interoperación en el ámbito del análisis automático morfosintáctico y sintáctico, que sirva de base para trabajos futuros.

4. ESTADO DEL ARTE

Conociendo las necesidades del software que se va a desarrollar, es necesario realizar un estudio y un análisis más profundos de los distintos campos y tecnologías que abarca el proyecto. Este capítulo resume (1) las características de la herramienta *FreeLing:* (2) sus limitaciones; (3) los problemas de interoperabilidad de las herramientas lingüísticas: (4) las posibles alternativas de implementación de este proyecto; (5) los requisitos propios de las anotaciones morfosintácticas y sintácticas normalizadas. El principal objetivo de este estudio es tomar las decisiones de diseño e implementación más adecuadas y oportunas para la consecución de los objetivos del proyecto.

4.1. ¿QUÉ ES FREELING?

FreeLing es una herramienta de análisis y etiquetado lingüístico. La versión actual, *FreeLing 3.0,* permite identificar el lenguaje al que pertenece una expresión lingüística, dividirla en oraciones, analizarla y etiquetarla morfosintácticamente, detectar y clasificar sus entidades con nombre (*named entities* en inglés), reconocer sus fechas, números, magnitudes físicas y porcentajes, analizar sus dependencias sintácticas, etiquetar su significado mediante *synsets de (Euro)WordNet*[5] y resolver sus correferencias. Además, en sus futuras versiones se esperan mejoras de rendimiento de las funciones existentes y la incorporación de nuevas características como desambiguación de oraciones, clasificación de documentos, etc. [4].

FreeLing, en resumen, es una aplicación de código abierto para el procesamiento automático del lenguaje natural, que proporciona una amplia gama de servicios de análisis lingüístico para una gran variedad de idiomas. *FreeLing* ofrece a los desarrolladores de aplicaciones de procesamiento del lenguaje natural funciones de

[5] Princeton University "About WordNet.": http://wordnet.princeton.edu/. Consultado en octubre de 2012.

Euro WordNet General Document: http://www.hum.uva.nl/-ewn. Consultado en octubre de 2012.

análisis y anotación lingüística de textos, con la consiguiente reducción del coste de construcción de dichas aplicaciones [4]. Además, está diseñado para su uso como librería externa en cualquier aplicación que requiera este tipo de servicios. Esta librería es personalizable y ampliable, y está fuertemente orientada al desarrollo de aplicaciones del mundo real en términos de velocidad y robustez. Asimismo, contiene un programa de ejemplo con una interfaz básica, que permite al usuario analizar archivos de texto desde la línea de comandos[6].

Por estos motivos, *FreeLing* es la herramienta con la que se va trabajar y para la que se va a generar la envoltura software descrita anteriormente, creándose así una nueva versión mejorada de *FreeLing*.

Hemos elegido esta herramienta porque es gratuita, de código abierto y permite a los desarrolladores utilizar sus recursos lingüísticos por defecto, como son los diccionarios, léxicos o gramáticas, para ampliarlos, adaptarlos o utilizarlos en dominios particulares. Dado que es una librería de código abierto, también es posible desarrollar otras nuevas funciones para idiomas específicos o necesidades especiales concretas.

4.2. LÍMITES DE FREELING

La herramienta *FreeLing* no fue concebida como una aplicación de propósito general, es decir, no está diseñada para ser fácil de usar, o para mostrar sus resultados mediante una interfaz amigable. Aun así, su API (*Application Programming Interface*, Interfaz de Programación de Aplicaciones) incluye una aplicación bastante completa, que permite a un usuario final, sin conocimientos de programación, obtener el análisis lingüístico de un cierto texto[7]. Aunque la interfaz básica ofrece un conjunto de opciones que cubren la mayor parte de sus funciones, es mucho más potente cuando

[6] FreeLing. http://nlp.lsi.upc.edu/FreeLing/index.php?option=com_content&task=view&id=12&Itemid=41. Consultado en octubre de 2012.

[7] FreeLing: límites. http://nlp.lsi.upc.edu/FreeLing/doc/userman/html/node4.html. Consultado en octubre de 2012.

se usa como API. De esta forma, se puede sacar mucho más partido a esta herramienta.

Los resultados de *FreeLing* son análisis lingüísticos que se almacenan en una estructura de datos. Cada aplicación que incorpore la interfaz de desarrollo *FreeLing* puede acceder a esos datos y procesar la información según lo considere necesario.
La principal limitación de la API es, sin duda, la falta de normalización de sus resultados, lo que dificulta su interoperación con otras herramientas. La aplicación de normas como MAF (para el análisis morfosintáctico), y SynAF (para el análisis sintáctico) a los resultados de la interfaz facilitaría, en gran medida, el uso y la integración de *FreeLing* con otras herramientas.

Además, el código disponible, aunque libre y gratuito, requiere de un proceso de instalación no trivial. Para completarlo, son necesarios muchas horas y conocimientos técnicos. En pocas palabras, se podría decir que el acceso a las funciones de *FreeLing* necesita una actualización urgente y adecuada que no requiera tantos recursos.

Este proyecto utilizará la API *FreeLing 3.0* de manera integrada, y se centrará en los dos aspectos básicos del análisis del lenguaje natural: el análisis morfológico y el análisis sintáctico. Es importante destacar una de las principales mejoras que se pretenden, que es la reimplementación de *FreeLing* como servicio web, lo que supondrá un acceso más fácil a sus funciones y, por lo tanto, dará lugar a una versión de *FreeLing* más interoperable con otras aplicaciones. Además, incluirá como valor añadido a la funcionalidad dada, la normalización de sus anotaciones de salida, mejorará la accesibilidad al servicio y, con esto, también se mejorará la interoperabilidad de sus resultados.

4.3. NECESIDAD DE INTEROPERABILIDAD Y PROBLEMAS EXISTENTES

Existen muchos conflictos y problemas que impiden que las herramientas y las anotaciones lingüísticas puedan interoperar y que, por lo general, provienen del enorme número de factores que se tienen en cuenta en su diseño y su desarrollo. Estos factores pueden ser tecnológicos (esto es informáticos), o teóricos, es decir, lingüísticos [2].

Hay varios factores teóricos que limitan la interoperabilidad de las herramientas y anotaciones lingüísticas. Para empezar, cuando dos herramientas de este estilo deben interoperar entre ellas, los conjuntos de fenómenos lingüísticos con los que tratan a menudo no coinciden. De hecho, por lo general, son disjuntos.

Por ejemplo, un analizador sintáctico se centrará en los fenómenos relacionados con la sintaxis (u orden relativo) de la entrada del texto recibida, mientras que un etiquetador semántico trabajará sobre el sentido de cada elemento léxico en este mismo texto [3].

Además, incluso cuando el mismo fenómeno lingüístico se analiza o anota con diferentes herramientas del mismo tipo, el resultado depende en gran medida del modelo lingüístico o la teoría que las guía. Por ejemplo, un analizador sintáctico desarrollado según una perspectiva funcional intentará caracterizar el rol que desempeña cada elemento de entrada dentro de su contexto (sujeto, objeto, etc). Por otro lado, un analizador de lenguaje natural desarrollado según una perspectiva estructuralista determinará principalmente las relaciones de constitución que existen entre estos mismos elementos [2].

Así pues, los principales problemas (teóricos o lingüísticos) que pueden causar conflictos al hacer que las herramientas y anotaciones lingüísticas interoperen son [3]:

- El conjunto de fenómenos y/o procesos que anota cada herramienta.
- El conjunto de etiquetas usadas para representar las categorías lingüísticas y los análisis resultantes. Estos *tagsets* o conjuntos de etiquetas difieren entre sí por los siguientes motivos:

1. Los aspectos de un mismo fenómeno que estudian las teorías con las que se interpretan los fenómenos lingüísticos difieren entre sí.
2. La precisión de las categorías utilizadas para su caracterización y/o anotación no es similar.
3. Las etiquetas que codifican una misma categoría son distintas en cada herramienta.

Por lo tanto, cuando se comparan dos herramientas lingüísticas habrá que analizar cada uno de estos factores tecnológicos y cuestiones teóricas para determinar su grado efectivo de interoperabilidad [3].

4.4. POSIBLES ALTERNATIVAS DE IMPLEMENTACIÓN

En este punto se muestran distintas tecnologías actuales orientadas a solucionar los problemas planteados de interoperabilidad y normalización. En el apartado 4.4.1 se explica la importancia de la web semántica y los distintos paradigmas de comunicación que hacen posible la interoperabilidad. En el apartado 4.4.2. se detalla el marco de anotación sintáctica y morfosintáctica en el cual se enmarcan los resultados normalizados de las salidas.

4.4.1. EL ROL DE LA WEB SEMÁNTICA Y LOS SERVICIOS WEB

Las tecnologías de la información y el acceso a la web se han hecho omnipresentes en nuestro día a día. Actualmente, los enfoques basados en la web se han convertido en el modo de trabajo por defecto de los procesos de negocio y de la automatización de diversas operaciones empresariales. Las diferentes comunidades verticales de mercado, tales como las telecomunicaciones, ciertos procesos de fabricación o las ciencias de la salud, hacen uso de sistemas de información basados en la web para optimizar sus procesos y obtener una ventaja competitiva en el mercado [5].

Es en este contexto en el que nació la Web Semántica, que es una extensión de la web actual en la que se dota a la información de un significado bien definido, lo que facilita que las personas y los ordenadores trabajen juntos [5]. La Web Semántica tiene como objetivo definir y analizar los datos que hay en la web de forma que puedan ser utilizados por las máquinas, no sólo para la visualización de información, sino también para la automatización, integración y reutilización de estos datos.

En consecuencia, el objetivo de la Web Semántica es tan amplio como el de la web: crear un medio universal para el intercambio de datos [5]. No obstante, en el caso de la Web Semántica, este objetivo es aún más ambicioso, pues busca que estos datos se representen en un formato computable, es decir, comprensible para los ordenadores (véase la Figura 1).

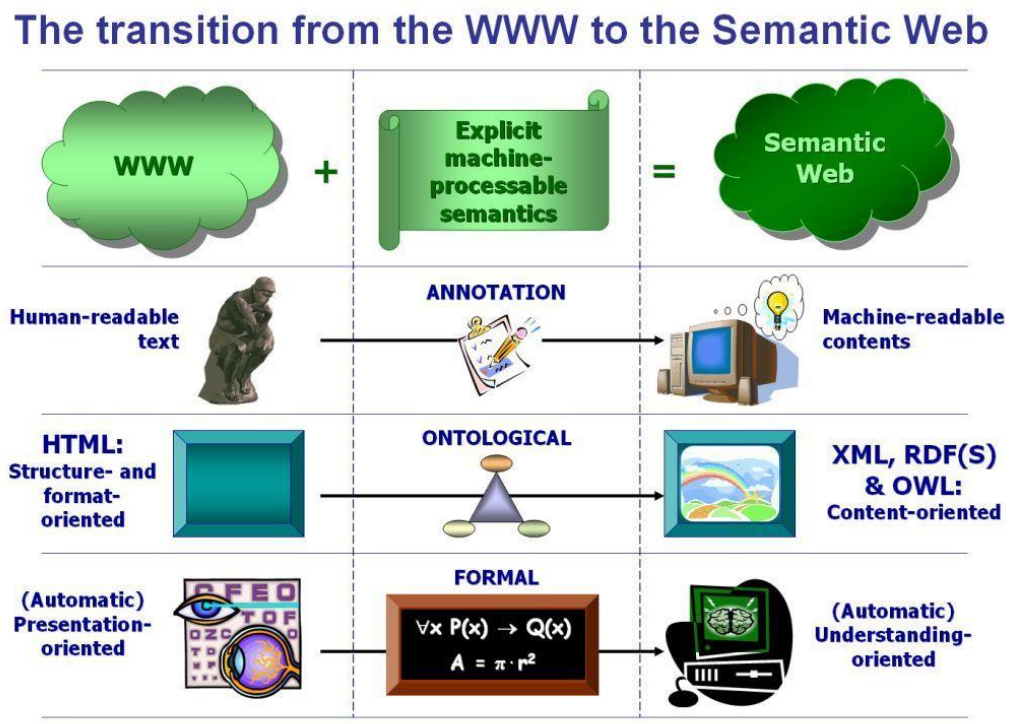


Figura 1: La Web Semántica[8].

[8]Pareja-Lora, A. *Anotación Semántica y Recuperación de Información*. http://www.slideshare.net/profesorescuelaveranoucm/fesabid-2007-apareja-lora-4710221) . Consultado en enero de 2013.

En efecto, la Web Semántica trata de explicitar formalmente el significado de los contenidos de los documentos en la red, y obtener así una red de contenidos comprensibles y procesables por los ordenadores.

La Web Semántica surgió para salvar la dificultad de encontrar e integrar la sobreabundancia de información y servicios distribuidos en la web. Los aspectos más importantes que están impulsando el desarrollo de la Web Semántica son los siguientes [5]:

- La reducción de los flujos de trabajo y del coste del análisis de los datos.
- La unificación del acceso a la información y a los servicios.
- La automatización de los procesos de publicación de datos, así como la interoperabilidad de los datos publicados.
- La escalabilidad y la interoperabilidad entre sistemas de información.
- El aumento de la colaboración entre los miembros de comunidades científicas, para el intercambio de información.
- La creación de vocabularios específicos para el etiquetado y el acceso a la información y para la interoperabilidad de los procesos.
- La generación metadatos, anotaciones y ontologías que faciliten la colaboración y el uso de la información.

La web sólo podrá alcanzar su pleno potencial si se convierte en un lugar en el que los datos puedan ser compartidos y procesados no sólo por personas, sino también por herramientas automatizadas. Los programas del futuro deben ser capaces de compartir y procesar los datos, incluso, cuando se hayan diseñado de forma totalmente independiente. Este es uno de los principales objetivos y la gran motivación que late tras la visión de la Web Semántica[9].

[9] W3C: *Semantic Web.* http://www.w3.org/2001/sw/. Consultado en octubre de 2012.

4.4.1.1. Paradigmas de comunicación

La Web Semántica no sería posible sin las redes de comunicación, que permiten a cualquier usuario conectado a las mismas acceder a un sitio web público sin esfuerzo, y de manera muy rápida y eficiente [5].

Esta facilidad de uso viene determinada por el formato normalizado de los datos que se publican en la web y por el protocolo de comunicación, también normalizado, que permite el acceso a los datos. Ambos se pueden implementar fácilmente en cualquier plataforma informática. Esta sencillez de implementación asegura el acceso universal a los datos, independientemente del equipamiento informático del que se disponga. Los servicios que sustentan la Web Semántica son los mecanismos existentes de comunicación y coordinación entre aplicaciones informáticas [5].

Se han desarrollado diferentes paradigmas de comunicación en función de sus características de comunicación: mediante conexión síncrona, asíncrona o a través de variables compartidas [5]:

- **Cliente/Servidor (C/S):** Este paradigma es uno de los más antiguos y se basa en un tipo de conexión síncrono. En este enfoque se distingue (1) un proveedor de servicios, denominado servidor; y (2) un consumidor del servicio, denominado cliente. El cliente y el servidor pueden estar, o no, en el mismo equipo. La comunicación se establece a través de una red local o de área extensa.

  El servidor define y especifica las posibles llamadas que un cliente puede hacer, así como el orden de las mismas. Además, define las estructuras de datos y su contenido.

  El cliente no tiene la capacidad de influir en ninguna de estas definiciones, tan sólo decide el momento en el que quiere entablar la conexión. Los clientes no pueden comunicarse entre sí, sino que toda la comunicación es bilateral entre un cliente y el servidor.

- **Teoría de Colas:** Este paradigma se ha hecho popular gracias a los sistemas de gestión de colas explícitas, como es el caso de los centros de atención al usuario. Las principales características que definen este tipo de comunicación son las siguientes [5]:

    I. Conexión asíncrona.

    II. Identificación del nivel óptimo de capacidad del sistema, lo que minimiza su coste.

    III. Evaluación de la capacidad del sistema que se estima necesaria y sus consecuencias en el coste total del mismo.

    IV. Establecimiento de un equilibrio entre coste y servicio.

  Los clientes que requieren un servicio entran al sistema uniéndose a una cola. En un determinado momento, se selecciona un miembro de la cola para ser servido, tras lo cual sale de la cola del sistema. Es importante tener en cuenta la disciplina de la cola, es decir, el orden en el que se seleccionan sus miembros para recibir el servicio. Por ejemplo, puede ser [5]:

    I. FIFO (*first in first out*: primero en entrar, primero en salir), según la cual se atiende primero al cliente que antes haya llegado.

    II. LIFO (*last in first out:* último en entrar, primero en salir), también conocida como pila, que consiste en atender primero al cliente que ha llegado el último.

    III. RSS (*random selection of service:* selección aleatoria del servicio) que selecciona los clientes de manera aleatoria, de acuerdo a algún procedimiento de prioridad, por ejemplo.

    IV. Procesador Compartido: sirve a los clientes de forma igualitaria. La capacidad del sistema se comparte entre los clientes y todos experimentan el mismo retraso.

- **Peer-to-Peer (P2P):** Este paradigma es muy similar al paradigma cliente/servidor. La conexión también es síncrona, pero la principal diferencia es que el servidor no es estacionario, es decir, no existe un servidor central al que se conectan todos los clientes. Las redes P2P están formadas por diversos nodos, que se comunican directamente entre sí. Cada nodo puede funcionar como cliente o como servidor, indistintamente, y es posible que un cliente se conecte al mismo tiempo con varios nodos, actuando con roles distintos en las diferentes conexiones [5].

- **Pizarra:** Es un espacio donde los datos pueden ser publicados, modificados o eliminados por un cliente. A priori, se desconoce el momento en el que se van a poner los datos a disposición de los usuarios, o cuánto tiempo va a permanecer la información allí. Los clientes pueden leer datos en cualquier momento y cuantas veces lo necesiten, y no hay limitación en el número de emisores y receptores de información [5].

- **Servicios web (WS, *Web Services*):** Nuevo paradigma de comunicación, formado por un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer servicios. Los proveedores ofrecen sus servicios como procedimientos remotos, y los usuarios solicitan un servicio llamando a estos procedimientos a través de la web. Estos servicios proporcionan mecanismos de comunicación **normalizados** entre diferentes aplicaciones, que interactúan entre sí para presentar información al usuario dinámicamente. Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia normalizada[10].

[10] Wikipedia: Servicio Web. http://es.wikipedia.org/wiki/Servicio_web. Consultado en febrero de 2013.

- **Transferencia de Estado Representacional (REST, *Representational State Transfer*):** Es una forma particular de comunicación, basada en el principio de que todos los datos, así como las operaciones con los mismos, se habilitan mediante URLs estáticas únicamente, y basada en el protocolo HTTP. Su idea fundamental es ver los datos y las operaciones como recursos identificables y las URLs como su mecanismo de identificación. Como tal, los recursos (datos u operaciones) se identifican mediante direcciones URL. La comunicación es síncrona y mediante el protocolo HTTP. Las URLs estáticas aseguran que se pueda acceder a los datos o a la función con la misma URL en cualquier momento. La comunicación es bilateral, ya que sólo puede haber dos participantes en una comunicación. Sin embargo, un servidor puede proporcionar respuestas a varios clientes. En este paradigma de comunicación no existe una definición explícita de los formatos de datos; todos los aspectos, incluyendo la estructura de la URL, se definen por el servidor de forma implícita [5].

- **Agentes:** es un concepto derivado del campo de la Inteligencia Artificial (AI). Los agentes son entidades autónomas que pueden comunicarse con otros agentes, no sólo comunicando datos, sino también solicitando explícitamente tareas específicas. El agente puede preguntar, ejecutar, delegar o rechazar tareas. En este sentido, existe una noción explícita de aceptación, así como de rechazo de tareas. Un agente puede comunicarse con cualquier número de agentes. Sin embargo, toda comunicación es bilateral, es decir, la comunicación es directa entre cada par de agentes. Los formatos de comunicación no están predefinidos, puede acordarse en cada conexión los formatos y contenidos de los datos que se desean utilizar.

- **Los espacios de tuplas (*Tuple Spaces*):** Es un paradigma muy similar al de las pizarras, sólo que en este caso las estructuras de datos están predefinidas como tuplas y todas las entidades que entran en comunicación tienen que seguir esta estructura. Este paradigma sigue el principio básico de variable compartida, donde el acceso al espacio de tuplas es concurrente y, sin embargo, cada individuo accede atómicamente por razones de coherencia. Los formatos no están predeterminados, y

se determinan y definen por ambas partes. En este enfoque, la comunicación puede ser bilateral y multilateral, ya que cualquier número de individuos puede comunicarse al mismo tiempo en distintos espacios de tuplas [5].

- **Co-ubicación:** Es un paradigma de comunicación en el cual el emisor y receptor comparten su código de comunicación, de manera que uno de los dos realiza únicamente invocaciones locales [5].

- **Computación de objetos distribuidos:** No es tanto un paradigma de comunicación, sino más bien un sistema de programación orientado a objetos que permite la comunicación a través de los objetos. Permite que dichos objetos, que se distribuyen a través de una red, puedan interoperar como un todo unificado. Estos objetos pueden estar distribuidos en diferentes ordenadores, y sin embargo parecen pertenecer a una aplicación. Tres de los paradigmas de computación de objetos distribuidos más populares son DCOM (Modelo de Objetos con Componentes distribuidos de Microsoft), CORBA (Arquitectura basada en un *broker* de peticiones comunes de OMG) y Java RMI (Invocación de métodos remotos de JavaSoft)"[11].

A continuación se muestra en la Tabla 1 un resumen comparativo de los paradigmas de comunicación anteriormente explicados:

[11] *A Detailed Comparison of CORBA, DCOM and Java/RMI.* http://my.execpc.com/~gopalan/misc/compare.html. Consultado en julio de 2013.

Tabla 1: Comparativa sobre paradigmas de comunicación.

PARADIGMA/COMUNICACIÓN	Bilateral	*Multilateral*	Síncrona	Asíncrona	Variables compartidas	Invocación independiente de la plataforma
C/S	X		X			
Teoría de colas	X		X			
P2P		X	X			
Pizarra		X		x		
Servicios web	X		X	x		X
REST						
Agentes						
Espacios de tuplas	X	X		X	X	
Co-ubicación	X		X			
Computación de objetos distribuidos	X	X	x			

4.4.1.2. Los servicios web en detalle

Tras analizar los distintos paradigmas de comunicación (véase el apartado anterior), se ha optado por los servicios web para la realización de la aplicación de recubrimiento y normalización de recursos lingüísticos. El motivo es que para dicha aplicación se necesita un tipo de comunicación síncrona, donde el cliente, en el momento en que lo desee puede acceder al servicio deseado. Además, como se observa en la Tabla 1, los servicios web permiten realizar invocaciones a los mismos desde cualquier plataforma, debido a que disponen de unos protocolos de comunicación **normalizados,** que facilitan el acceso y la interoperabilidad en la web, que es uno de los objetivos deseados.

A continuación se explica con más detalle en qué consisten los servicios web, sus ventajas e inconvenientes.

5.4.1.1.1. Concepto de servicio web.

Existen múltiples definiciones de servicio web (en inglés, *Web Service*), lo que muestra su complejidad, pues deben englobar todo lo que son e implican. Una posibilidad sería hablar de ellos como "un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la web"[12]. Otra definición muy clara sería la siguiente: un servicio web es una tecnología que utiliza un conjunto de protocolos y normas que sirven para intercambiar datos entre aplicaciones.

Los servicios web surgieron de la necesidad de comunicación entre aplicaciones: existían muchas aplicaciones, pero eran incapaces de comunicarse entre sí. Esto motivó la creación de software (el servicio) que envolviera dichas aplicaciones para poder ser invocadas por cualquier cliente o aplicación, sin depender de la tecnología utilizada. "Así, las distintas aplicaciones, aún desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de normas abiertas. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles que definan estas normas más detalladamente. En definitiva, un servicio web es una máquina que atiende las peticiones de los clientes web y les envía los recursos solicitados"[13].

---

[12] W3C: *Web Service.* http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb. Consultado en octubre de 2012.

[13] Wikipedia: Web Services. http://es.wikipedia.org/wiki/Servicio_web. Consultado en octubre de 2012.

5.4.1.1.2. Para qué sirven los servicios web.

Los servicios web proporcionan mecanismos de comunicación normalizados entre diferentes aplicaciones, que interactúan entre sí para presentar al usuario información generada dinámicamente. Para hacer que estas aplicaciones sean extensibles e interoperables, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia también normalizada.

La Figura 2 muestra cómo interactúa un conjunto de servicios web:

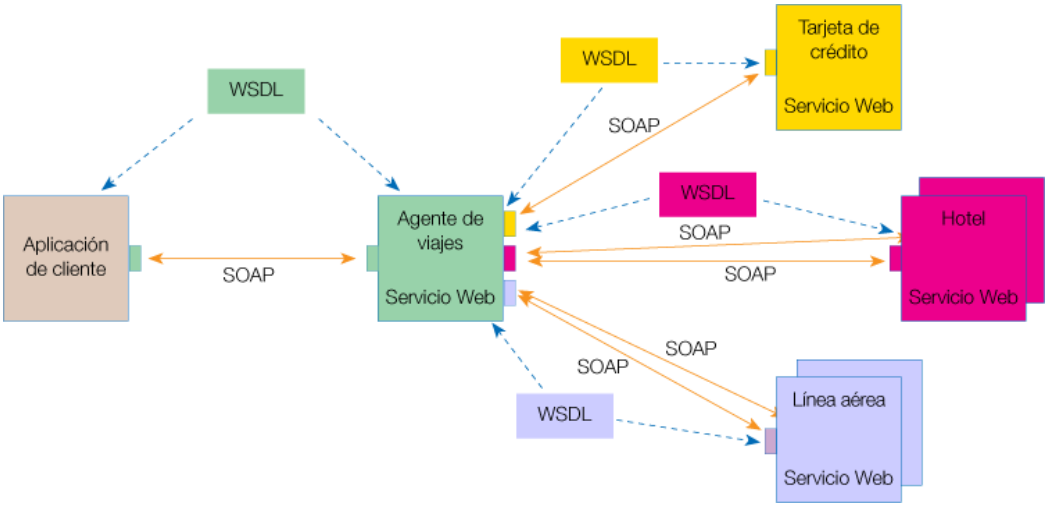


Figura 2: Los Servicios web en funcionamiento[14].

Según el ejemplo de la Figura 2, un usuario (que juega el papel de cliente dentro de los servicios web), solicita información sobre un viaje que desea realizar. Para ello, a través de una aplicación, hace una petición a una agencia de viajes que ofrece sus servicios a través de Internet. La agencia de viajes ofrecerá a su cliente (usuario) la

[14] W3C: Servicios Web. http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb. Consultado en octubre de 2012.

información requerida. Para proporcionar al cliente la información que necesita, esta agencia de viajes solicita a su vez información a otros recursos (otros servicios web) en relación con el hotel y la línea aérea. La agencia de viajes obtendrá información de estos recursos, lo que la convierte a su vez en cliente de esos otros servicios web, que le van a proporcionar la información solicitada sobre el hotel y la línea aérea. Por último, el usuario realizará el pago del viaje a través de la agencia de viajes, que servirá de intermediario entre el usuario y el servicio web que gestionará el pago.

Las alternativa a esta secuenciación suponía que, o bien el cliente o bien la agencia, debía realizar cada parte del proceso de manera independiente, no automatizada. Esto implicaría una comunicación distinta para resolver cada servicio, lo que comporta un mayor gasto de tiempo y recursos.

En todo este proceso intervienen una serie de tecnologías que hacen posible este intercambio de información. Por un lado, estaría SOAP (*Simple Object Access Protocol*: Protocolo Simple de Acceso a Objetos). Se trata de un protocolo basado en XML, que permite la interacción entre varios dispositivos y que tiene la capacidad de transmitir información compleja. Los datos pueden ser transmitidos a través de HTTP, SMTP, etc. SOAP especifica el formato de los mensajes. El mensaje SOAP está compuesto por un *envelope* (sobre), cuya estructura está formada por los siguientes elementos: *header* (cabecera) y *body* (cuerpo). Y por último, WSDL (*Web Services Description Language*: Lenguaje de Descripción de Servicios Web), permite que un servicio y un cliente establezcan un acuerdo en lo que se refiere a los detalles de transporte de mensajes y su contenido. WSDL especifica la sintaxis y los mecanismos de intercambio de mensajes.

5.4.1.1.3. Ventajas de los servicios web

El uso de servicios web tiene las siguientes ventajas[15]:

- Permiten que las aplicaciones informáticas sean independientes de la tecnología utilizada, es decir, del lenguaje de programación o de las plataformas sobre las que se instalen.
- Los servicios web se basan en normas y protocolos que usan XML, lo que les da un carácter universal en su uso, da mayor facilidad de acceso a su contenido y permite entender mejor su funcionamiento.
- Permiten que servicios y aplicaciones de diferentes compañías, ubicadas en diferentes lugares geográficos, puedan ser combinados fácilmente para proporcionar servicios integrados.

5.4.1.1.4. Inconvenientes de los servicios web

Sin embargo, los servicios web también presentan los siguientes inconvenientes[16]:

- No puede compararse su grado de desarrollo para realizar transacciones con las reglas de computación distribuida, como CORBA.
- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI, CORBA o DCOM. Es uno de los inconvenientes derivados de adoptar un formato basado en XML. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.
- Al apoyarse en HTTP, pueden esquivar ciertas medidas de seguridad basadas en cortafuegos (*firewall)*.

[15] Wikipedia: Web Services. http://es.wikipedia.org/wiki/Servicio_web. Consultado en octubre de 2012.

[16] Véase el comentario a la nota pie anterior.

4.4.2. EL ROL DE LA NORMALIZACIÓN

Cuando varias herramientas lingüísticas anotan o analizan una misma entrada, pueden producir resultados dispares y, aun así, correctos. Esto es posible porque o bien (a) prestan atención a distintos conjuntos de fenómenos lingüísticos, o (b) interpretan esta entrada de acuerdo con una perspectiva o una teoría lingüística distinta. Esta diferencia también se manifiesta en las salidas obtenidas, en función de los factores tecnológicos tenidos en cuenta, el conjunto de etiquetas definido, su formato y/o su estructura [2].

Los principales factores tecnológicos que dificultan la interoperación de las herramientas lingüísticas y la normalización de sus resultados están asociados, principalmente, con los formatos y esquemas de sus entradas y salidas [3]. Respecto a los formatos de sus entradas, estos son muy limitados. En la mayoría de los casos, se admite únicamente un tipo de entrada y, en muchas ocasiones, con esquemas muy particulares, que no responden a normalización alguna. Respecto a los formatos de sus salidas, cada herramienta se suele ajustar a un esquema concreto, hecho a medida, y en la mayoría de los casos, estos esquemas, su conjunto de etiquetas, sus formatos y su estructura tampoco siguen ningún tipo de norma, directriz o formato estándar [2].

En este contexto tenemos que contar con ISO (*International Organization for Standardization:* Organización internacional de Normalización) que es el organismo encargado de promover el desarrollo de normas, siendo su función principal la de desarrollar normas de productos y seguridad para las empresas u organizaciones (públicas o privadas) a nivel internacional. La ISO es una red de institutos de normas nacionales de 164 países, sobre la base de un miembro por país, con una Secretaría Central en Ginebra (Suiza) que coordina el sistema.

La misión del Comité TC 37 de ISO es la estandarización de los principios, métodos y aplicaciones relacionadas con la terminología, las industrias de la lengua, así como los diferentes recursos terminológicos y de organización del conocimiento, dentro de los diversos contextos de comunicación multilingüe y diversidad cultural. Este comité

cuenta con 4 subcomités y diversos grupos de trabajo dentro de cada subcomité. Uno de los subcomités es ISO TC37/SC4, que es el encargado de la gestión de recursos lingüísticos[17].

4.4.2.1. MARCO DE ANOTACIÓN MORFOSINTÁCTICA (MAF)

Una de las normas de ISO TC37/SC4 es ISO 24611:2012, el marco de anotación morfosintáctico, MAF (*Morpho-Syntactic Annotation Framework*), que trata de (1) cómo delimitar las secuencias de elementos de la entrada que pueden ser identificadas como unidades morfosintácticas, y (2) cómo anotar estas unidades con sus categorías y/o propiedades lingüísticas correspondientes [6].
Dentro del ámbito del análisis del lenguaje natural, las anotaciones morfosintácticas proporcionan información lingüística sobre los componentes de cada oración de un texto o una transcripción [6].

Las unidades mínimas no pueden ser divididas en subcomponentes, sin embargo pueden dividirse en unidades más pequeñas con propiedades morfológicas y fonológicas. Además, las unidades morfosintácticas se pueden anidar para formar unidades más grandes, como palabras compuestas, que actúan como unidades elementales para el nivel de análisis sintáctico. El límite exacto entre la morfosintaxis y el análisis sintáctico es a veces difícil de definir [6].

La norma MAF da tratamiento y definición a las distintas categorías gramaticales de cada elemento de la oración. No obstante, es difícil dar una definición exacta y precisa de lo que las anotaciones morfosintácticas cubren, ya que están estrechamente relacionadas con muchas otras propiedades lingüísticas de cada lengua y del contexto [6].

[17] ISO TC 37: http://www.iso.org/iso/home/standards_development/list_of_iso_technical_committees/iso_technical_committee.htm?commid=48104. Consultado en marzo de 2013.

Esta norma internacional se basa en un metamodelo que configura la estructura de las anotaciones morfosintácticas basándose en sus dos elementos principales: *Token* y *WordForm.*

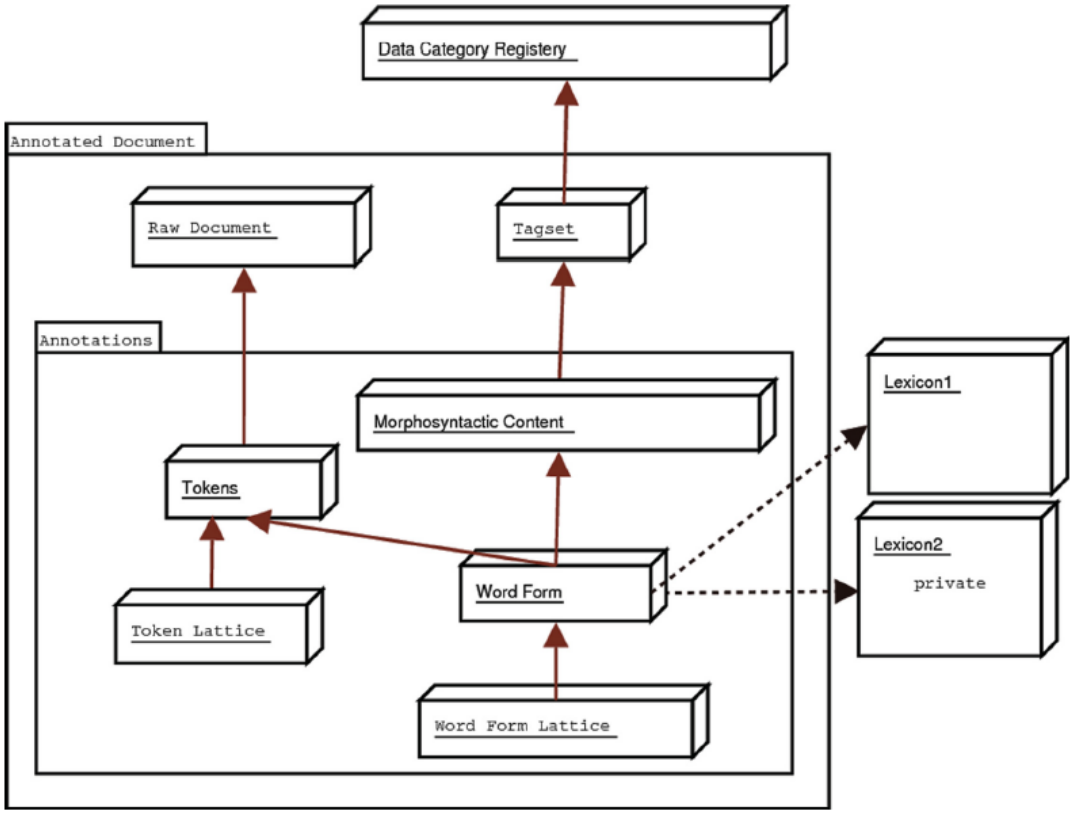


Figura 3: Elementos básicos de la anotación morfosintáctica [6].

El elemento *Token* se utiliza para representar los segmentos del documento original o texto, que respeten las reglas ortográficas, morfológicas y fonéticas del idioma. Los *Token* son las unidades mínimas de representación de la información. Entre cada par de etiquetas "`<token> </token>`" están incluidos los segmentos de la oración que se corresponden con partes que tienen un significado atómico e independiente.

Por ejemplo, la oración: “Díselo al médico” podría descomponerse en lo siguiente:

|Dí|se|lo|a|l|médico|

Cada una de estas unidades posee un significado independiente, que es preciso anotar.

Los *WordForm* son unidades lingüísticas compuestas por segmentos identificados por elementos `<token>`. Cada uno de estos *WordForm* puede contener uno o más *Token*. Por ejemplo, para la contracción “del”, compuesta por “de” + “el” su conjunto de *Tokens* será:

```
<token xml:id=”t1”> de </token>
<token xml:id=”t2” join=”left”> l </token>
```

El atributo *xml:id* identifica individualmente cada *Token*, en este caso a t1 y t2, de tal manera que se pueda hacer referencia a ellos. Por otro lado, el atributo *join=”left”* lo que representa es la relación que tiene dicho *Token* con el *Token* inmediatamente contiguo a su izquierda. Por otra parte, se deben omitir redundancias, si en un texto se encuentra la contracción “del” no se muestran en etiquetas *Token* “de” y “el” sino en etiquetas “de” y “l”. De esta manera se refleja con exactitud la composición del texto con su normalización etiquetada.

El número de *WordForms* para el ejemplo serían tres, uno por cada *Token:*

```
<wordForm xml:id="wordForm1" lemma="de" token="t1"/>
        <fs>
            <f name="pos">
                <symbol value="AP"/>
                  <!-- Adposición (Preposición) -->
            </f>
        </fs>
</wordForm>

<wordForm xml:id="wordForm2" lemma="el" tokens="t2"/>

        <fs>
            <f name="pos">
                <symbol value="ATDMS"/>
                  <!— Articulo Definido, Masculino, Singular -->
            </f>
        </fs>
</wordForm>
```

Además, se necesita un *WordForm* para la composición de los dos *Tokens* anteriores “de” y “l” puesto que, el conjunto de ambos es una palabra en sí misma: “del”, y por tanto también debe estar encapsulado en un elemento `<WordForm>`:

```
<wordForm xml:id="wordForm5" tokens="t4 t5"/>
```

El atributo *token* hace referencia al *Token* o *Tokens* que componen el *WordForm.* Y, por último, el atributo *value* de la etiqueta `<symbol>` muestra el valor *tag* del *Token* en forma de abreviatura y su significado descrito en un comentario.

4.4.2.2. MARCO DE ANOTACIÓN SINTÁCTICA (SynAF)

En 2010, el subcomité ISO TC37/SC4 publicó la norma 24615 (SynAF), que proporciona un modelo de referencia para la representación de anotaciones sintácticas. Esta norma se asienta sobre propuestas previas para la representación de la información sintáctica y proporciona un marco global que permita [7]:

- Representación de las anotaciones sintácticas y su estructura a través de la etiqueta <corpus> definida en la norma.
- Cubrir la representación tanto de dependencias como de relaciones de constitución sintáctica.
- Ser lo suficientemente flexible como para hacer frente a una variedad de idiomas, ya que, se basa en un metamodelo abierto, el cual, puede ser parametrizado por categorías de datos específicos.

Sin embargo, la norma SynAF no contiene ninguna serialización XML específica, y esto la hace insuficiente para garantizar su interoperabilidad. Esta es la razón por la que el subcomité ISO TC 37/SC 4 inició un nuevo proyecto con el fin de proporcionar una parte adicional a SynAF sobre la base del formato Tiger-XLM [8]. Este formato ha sido ligeramente adaptado, para cumplir con los recientes desarrollos la tecnología XML e integrar algunas de las características necesarias para abarcar completamente las anotaciones basadas en dependencias. El resultado de esta adaptación se denomina <tiger2/> [9].

El estándar ISO 24615 (SynAF) proporciona un metamodelo de alto nivel para la representación de la anotación sintáctica de datos lingüísticos, con el objetivo de favorecer la interoperabilidad entre los distintos recursos y componentes del procesamiento del lenguaje natural. SynAF tiene como objetivo representar las dos formas principales de anotación sintáctica en la actualidad [9]:

a) La de relaciones de constitución lingüística. Por ejemplo, en la oración "El presidente de España se llama Mariano Rajoy." encontramos el sintagma nominal (SN) "El presidente", que está constituido por el artículo "el" y el nombre "presidente".
b) Las relaciones de dependencia sintáctica, tales como las relaciones existentes entre categorías del mismo tipo (por ejemplo, las relaciones entre los sustantivos en aposiciones o, más concretamente, la relación que existe entre "Mariano" y "Rajoy" en "El presidente de España se llama Mariano Rajoy." - modificador adjetivo) o entre los núcleos y sus modificadores (como la dependencia que hay entre "presidente" y "llama" en esa misma oración, pues "presidente" es el sujeto de "llama").

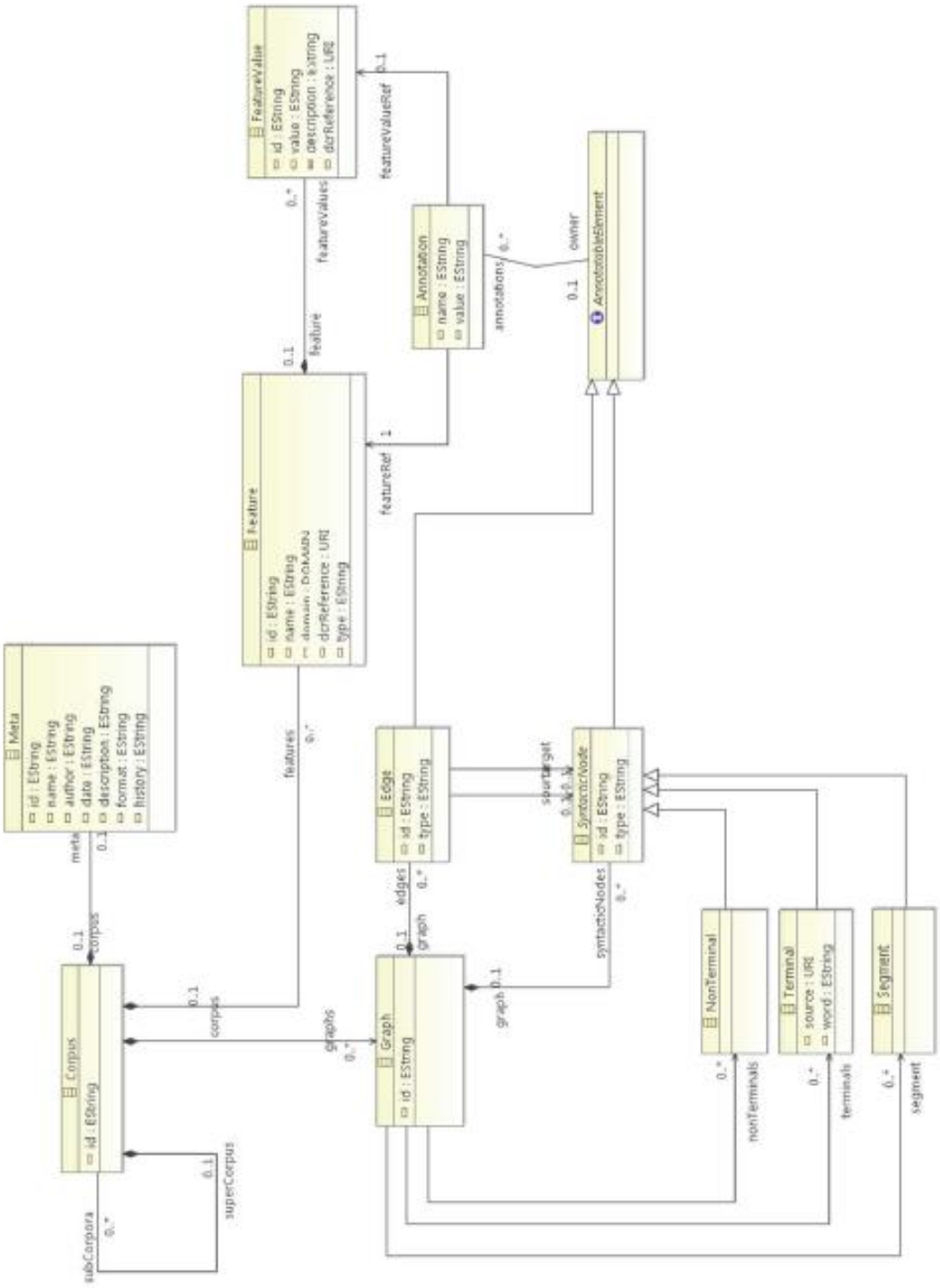


Figura 4: Diagrama SynAF [9].

Como consecuencia, las anotaciones sintácticas deben cumplir con una estrategia de anotación multicapa, que interrelacione las anotaciones sintácticas de constitución y dependencias, en forma conjunta, como se indica en el metamodelo SynAF.

El metamodelo de <tiger2/> se compone de tres partes:

1) La estructura organizativa del corpus y sus metadatos asociados.

2) La definición del conjunto de etiquetas que se usan para su anotación.

3) El gráfico con las anotaciones lingüísticas, encapsuladas en un elemento <graph>.

La estructura organizativa de los corpus se representa mediante el elemento recursivo <corpus> y sus correspondientes metadatos, representados mediante el elemento <meta> [9].

La definición del conjunto de etiquetas o *tagset* se encapsulan en la etiqueta <annotation> compuesta de diferentes <feature>. Esta última etiqueta enumera el dominio de valores posibles para cada elemento de la estructura. Los elementos de la estructura pueden ser terminales, no terminales o aristas [9].

```
<head>
      <annotation>
            <feature xml:id="f1" name="pos" domain="t" />
            <feature xml:id="f2" name="lemma" domain="t" />
            <feature xml:id="f3" name="cat" domain="nt"/>
            <feature xml:id="f4" name="label" domain="edge"/>
      </annotation>
 </head>
```

El grafo de la anotación lingüística define un conjunto de elementos que contienen los datos primarios y la estructura de anotación que atañe a los datos primarios. Está formado por el elemento <graph> y representa un conjunto de nodos relacionados entre sí y sus aristas o edges. No se especifica qué partes de un texto primario están cubiertos por una sola gráfica: un gráfico puede cubrir por ejemplo, una frase, una subfrase, un capítulo o un texto entero.

```
<s xml:id=”s1”>
      <graph xml:id="s1_g1">
            <terminals>
                  <t xml:id="s1_t1" corresp="example.maf#wordform_1" />
                  <t xml:id="s2_t2" corresp="example.maf#wordform_2"/>
            </terminals>
            <nonterminals>
                  <nt xml:id="s1_nt1">
```

```
                        <edge xml:id="s1_e1" target="#s1_t1" />
                        <edge xml:id="s1_e2" target="#s1_t2" />
                    </nt>
                </nonterminals>
            </graph>
</s>
```

Un segmento del texto está representado por el elemento <s>. Este debe contener uno o más elementos <graph>, expresando posibles subgrafos o anotaciones múltiples dentro de un segmento.

Dentro del elemento <graph> se sitúan las etiquetas que representan los nodos terminales: <terminals>, y los nodos no terminales: <nonterminals>. Cada uno de ellos contendrá respectivamente los elementos <t> y <nt>:

1. El elemento <t> representa un nodo terminal y se anidan en un elemento <terminals>. Un nodo terminal <t> constituye el punto de referencia para los datos primarios. Esta referencia puede ser directa a la parte de un texto que está reflejado en el mismo documento <tiger2/> o puede ser indirecta, es decir, fuera del modelo <tiger2/>, como por ejemplo, a un elemento Wordform contenido en un archivo MAF utilizando el atributo *corresp*.

2. El elemento <nt> representa un nodo no terminal y está anidado en el elemento <nonterminals>. Un nodo no terminal es un nodo interno, en referencia directa o indirecta a un nodo terminal en el mismo documento <tiger2/>. Los <nt> se anotan con categorías sintácticas válidas de los niveles sintagmático no posicional y oracional.

3. El elemento <edge> representa una relación entre dos nodos tanto terminales como no terminales y viene a ser la arista de unión. La etiqueta <edge> puede ser elemento secundario de un <t> o <nt>. El objetivo está dado por el atributo target y debe hacer referencia a un nodo dentro del mismo documento <tiger2/>.

5. DESARROLLO

El propósito de este capítulo es describir todas las fases de desarrollo del proyecto. La estructura es la siguiente: en primer lugar, la sección 5.1. 'Especificación de Requisitos'; en segundo lugar, la sección 5.2. 'Análisis y Diseño'; seguidamente, la sección 5.3. 'Implementación' y finalmente la sección 5.4. 'Pruebas'.

5.1. ESPECIFICACIÓN DE REQUISITOS (ERS)

En esta sección se explicarán y analizarán los requisitos de este proyecto de fin de grado, desarrollado en la Facultad de Informática (FDI) de la Universidad Complutense de Madrid (UCM) durante el curso 2013-2014. Se ha adoptado para ello la guía de requisitos del software del IEEE (Std. 830-1993) [10].
La estructura del resto de la sección se divide en tres epígrafes: el epígrafe 5.1.1. 'Introducción', el epígrafe 5.1.2.'Descripción General' y el epígrafe 5.1.3 'Requisitos Específicos'.

5.1.1. INTRODUCCION

Este epígrafe se divide en cinco puntos, en los cuales se detalla el propósito, alcance, definiciones y referencias utilizadas en este documento de ERS. En el último punto de este epígrafe se presenta una visión general del documento.

5.1.1.1. PROPÓSITO

La presente Especificación de Requisitos del Software (ERS) tiene como propósito definir de manera clara y precisa las funciones y restricciones que tendrá nuestro sistema, y sirve también como guía de desarrollo del software. Por tanto, debe dar a conocer el funcionamiento del sistema de manera general.

Esta especificación establece los objetivos de nuestro proyecto de fin de grado y los criterios de constatación de su consecución. Estará sujeto a revisiones hasta obtener la aprobación formal del sistema por parte del director del proyecto. En cuanto esto

ocurra, el documento funcionará como base para el desarrollo y construcción de la aplicación.

5.1.1.2. ALCANCE

El proyecto producirá el software "RNF"- Recubrimiento y Normalización de *FreeLing*. RNF proporcionará una capa o interfaz normalizada de acuerdo con las normas ISO/MAF [6] e ISO/SynAF [7] a la aplicación de análisis del lenguaje natural *FreeLing,* de tal manera que retorne sus salidas en archivos XML, en un formato normalizado según las normas mencionadas.

Por tanto, el desarrollo de RNF tomará una revisión reciente de *FreeLing*, que será modificada para crear una nueva versión que cumpla las normas indicadas. Esta nueva versión será realizada mediante un servicio web, que permanecerá operativo en un servidor físico accesible a cualquier usuario interesado en realizar análisis de textos escritos en español.

Criterios de éxito:

1. Entera satisfacción por parte del director del proyecto.
2. Facilitar al usuario toda la información necesaria para hacer un uso eficaz y eficiente de la aplicación
3. Cumplir todos los objetivos del proyecto, evitando todos los posibles fallos.

5.1.1.3. DEFINICIONES, ACRÓNIMOS Y ABREVIATURAS

- API: *Application Programming Interface* (Interfaz de programación de aplicaciones).
- ERS: Especificación de requisitos del software.
- HW: Hardware.
- IDE: Integrated development environment (Entorno de desarrollo integrado).

- IEEE: *Institute of Electrical and Electronics Engineers* (Instituto de Ingenieros Eléctricos y Electrónicos).
- ISO: *International Organization for Standardization* (Organización Internacional para la Normalización).
- LAF: *Linguistic Annotation Framework* (Marco de Anotación Lingüística).
- MAF: *Morpho-Syntactic Annotation Framework* (Marco de Anotación Morfo-Sintáctica).
- PoS: *Part Of Speech* (etiquetado gramatical).
- RNF: Recubrimiento y Normalización de *FreeLing*.
- SLE: Sistema de Lectura de Etiquetado.
- SOAP: *Simple Object Access Protocol* (Protocolo de Acceso a Objetos Simples).
- SW: Software.
- SynAF: *Syntactic Annotation Framework* (Marco de Anotación Sintáctica).
- UCM: Universidad Complutense de Madrid.
- UDDI: *Universal Description, Discovery and Integration* (Integración, Descubrimiento y Descripción Universal).
- UNED: Universidad Nacional de Educación a Distancia.
- Usuarios/Clientes: Hace alusión a las personas que interactúan con el servicio web.
- WSDL: *Web Services Description Language* (Lenguaje de Descripción de Servicios Web).
- XML: *eXtensible Markup Language* (Lenguaje eXtensible de Marcado).

5.1.1.4. REFERENCIAS

- IEEE Recommended Practice for Software Requirements Specification. ANSI/IEEE std. 830, 1998.
- ISO 24615 - Language resource management - Syntactic annotation framework (SynAF), 2010.

- ISO 24615 - Language resource management - Syntactic annotation framework (SynAF Part 2) - <tiger2/>, 2012.
- ISO 24611 - Language resource management - Morpho-syntactic annotation framework (MAF), 2012.
- ISO 12620 - Data Category Registry (DCR), 2009.

5.1.1.5. VISIÓN GENERAL DEL DOCUMENTO

Dentro del capítulo 5. 'Desarrollo', esta sección sobre ERS describe los requisitos del software que se va a construir. En las siguientes secciones: 5.2. 'Análisis y Diseño' y 5.3. 'Implementación', se profundizará con más detalle en las características de la aplicación.

Dentro de la ERS, se incluyen tres epígrafes:

- El presente epígrafe, presenta una introducción a la ERS, que contiene el propósito general de la misma, el ámbito del sistema, los acrónimos y las referencias utilizadas y una visión general de este.

- El epígrafe 5.1.2. 'Descripción General' presenta la descripción global del ERS, la cual incluye las perspectivas, funciones, características y restricciones del producto.

- El epígrafe 5.1.3. 'Requisitos Específicos' presenta los requisitos funcionales y no funcionales de la aplicación, detallando la utilidad de la aplicación.

5.1.2. DESCRIPCIÓN GENERAL

Existen factores o causas generales que afectan de manera continua al ciclo de vida del producto. En este epígrafe se identifican estos elementos, que constituyen el contexto de desarrollo del sistema.

5.1.2.1. PERSPECTIVA DEL PRODUCTO

Contexto de implantación:

Este proyecto consiste en una aplicación en forma de servicio web que tiene como función el análisis morfosintáctico y sintáctico en el idioma español, que son dos tareas básicas del procesamiento del lenguaje natural. El servicio web devolverá archivos XML normalizados, los archivos generados dependerán de la elección del usuario que solicite la llamada al servicio.

Interfaces de sistema:

El servicio web usará el código de *FreeLing* para proporcionar una salida normalizada de sus anotaciones al usuario. Los principales módulos que habrá que integrar son los siguientes:

- *Tokenizer*. Tokenizador del texto "en crudo". Realizará la preparación y segmentación del texto de entrada.
- *Morphological analyzer*. Análisis morfológico de la entrada.
- *PoS Tagger*. Etiquetado gramatical de las partes atómicas de la oración.
- *Chart Parser*. Analizador sintáctico, mostrado de manera gráfica.
- *Dependency Parser*. Analizador de dependencias sintácticas, mostrado de manera gráfica.

Interfaces de usuario:

El objetivo es proporcionar una interfaz para la anotación de textos en los niveles reseñados, que genere y sirva de puente entre las salidas de *FreeLing* y su serialización de acuerdo con ISO/MAF [6] e ISO/SynAF [7].

La interfaz ofrecida es el propio servicio web. Dicho servicio web será consumido por cualquier cliente que quiera utilizar el servicio. Como demostración de su utilidad, se dejará desplegado un cliente web que consuma el servicio disponible en el servidor de la UNED y/o UCM. El cliente tendrá dos opciones de análisis, que se realizarán mediante intercambios de peticiones o solicitudes al servicio web haciendo uso del protocolo WDSL.

Interfaces hardware:

No se necesitan interfaces adicionales a las proporcionadas por un PC convencional.

Interfaces software:

El equipo que provea de este servicio web necesitará tener cinco interfaces software disponibles:

1.- Como sistema operativo Ubuntu 12.04 o una versión compatible.

2.- *Freeling* correctamente instalado.

3.- Un servidor de aplicaciones donde desplegar el servicio web.

4.- La máquina virtual de Java instalada.

5.- Un navegador web con conexión a internet.

Interfaces de comunicación:

Es un servicio web que necesita de conexión a internet para poder recibir solicitudes del cliente para ejecutar el servicio y responder a dichas peticiones. El proceso se realiza en el servidor, el cliente solo debe proporcionar el texto mediante una solicitud *WSDL*.

Memoria:

Se trabajará con las memorias que puede proporcionar cualquier PC actual. Simplemente necesitamos poder albergar las informaciones relativas a las interfaces software necesarias para desplegar el servicio web. Para implantar el

servicio en el servidor proporcionado por la UNED y/o UCM que dispone previamente del servidor Apache Tomcat 6, se necesita 135 MB disponibles para instalar FreeLing y 32 MB para la instalación de Java 6.

En cuanto a la memoria secundaria, no es necesario ningún requisito adicional a lo proporcionado por cualquier ordenador actual.

Operaciones:

La aplicación contará con un único nivel de uso (acceso total) y no contará con ningún sistema de *backup*.

Instalaciones particulares:

Una vez codificado el servicio web, las primeras pruebas se ha realizado en el sistema operativo Ubuntu 12.04 con el servidor Glass Fish 3.1.2, incluido en el entorno de desarrollo de programación. Tras la depuración del servicio web y la creación del cliente web que consume el servicio, se realizó una segunda instalación en una máquina virtual, en esta ocasión con la versión de Ubuntu 11.04, y con Apache Tomcat 6 como servidor de Aplicaciones.

5.1.2.2. FUNCIONES DEL PRODUCTO

El objetivo de este desarrollo es procesar un texto dado en un formato específico y, por tanto, requiere de varios submódulos para procesar y extraer el resultado que se desea normalizar a través de la aplicación de nuestro servicio web.

Por consiguiente, para la realización de este cometido se deberá modularizar y definir las principales funciones generales del desarrollo. Cabe mencionar que los módulos resultantes serán contenedores de partes más pormenorizadas, que se encargarán de las funciones más atómicas, que definiremos en el diseño. Esto quiere decir que,

adicionalmente, los módulos contendrán las interfaces de los submódulos que necesitemos de *FreeLing*, al cual llamará, del cual recogerá sus resultados y los transformará para dar una salida normalizada.

De esta manera, las funciones principales del servicio web serán dos:

- Subsistema de Análisis Morfológico (SMAF): Módulo que contendrá el código desarrollado para conseguir que un documento XML, generado a partir de un análisis de un texto dado, sea normalizado debidamente según la norma ISO/MAF [6].
- Subsistema de Análisis Sintáctico (SSynAF): Módulo que contendrá el código desarrollado para transformar a un documento XML, generado a partir de un análisis de un texto de entrada, debidamente normalizado según la norma ISO/SynAF [7].

5.1.2.3. CARACTERÍSTICAS DE LOS USUARIOS

Los usuarios que pueden acceder a nuestro servicio web son exclusivamente aplicaciones clientes que consuman el servicio. Estos clientes pueden estar programados en cualquier lenguaje de programación sin dependencias a la plataforma tecnológica utilizada. En definitiva, cualquier clase de cliente puede llegar a obtener la UDDI y hacer uso de las funciones de RNF publicadas.

Por otra parte, para no tener que programar un cliente para probar el servicio, junto con el servicio web se despliega un cliente web que puede ser accesible por cualquier usuario, ingresando únicamente la URL del servicio.

5.1.2.4. RESTRICCIONES

A continuación se detallan las restricciones que se han tenido en cuenta a la hora de desarrollar nuestro proyecto:

- La aplicación estará alojada en la plataforma *Linux Ubuntu 12.04* o en otra plataforma compatible con esta.
- Estará realizado en forma de servicio web con el servidor de aplicaciones *Glass Fish* 3.1.2 y Apache Tomcat 6.
- Debe soportar el acceso de un gran número de usuarios en el sistema sin que se ocasionen problemas de concurrencia.
- Al ser una aplicación disponible para una gran cantidad de clientes, se deberá minimizar la posibilidad de interrupciones y fallos de nuestro servicio.
- Se utilizará Java como lenguaje de programación para su implementación, así como el *IDE Netbeans* (versión para L*inux*).
- Limitaciones HW: Tendremos en cuenta las limitaciones HW que el servidor de la UNED y/o de la UCM proporcionarán para la implantación del sistema y en los cuales se montará el servicio web.
- Políticas de regulación: La presente documentación se ceñirá al estándar 830 del IEEE para especificar los requisitos SW, así como los derivados de las normas ISO/MAF [6] e ISO/SynAF [7] antes mencionadas.
- Se deberá usar la API de *FreeLing* generada para Java y montar los sub-módulos recogiendo sus funcionalidades.
- Criticidad de la aplicación: las operaciones que se realicen en el Sistema no deberá exceder el tiempo de 60 segundos por consulta de un texto medio de 100 palabras.
- El texto dado no debe tener faltas ortográficas y respetar puntos y comas.

- El texto debe estar en idioma español. La finalización del texto introducido debe terminar obligatoriamente en un punto.

5.1.2.5. SUPUESTOS Y DEPENDENCIAS

Para el despliegue del servicio web se da por supuesto que se dispone del hardware y software necesario, explicado en este documento, y necesario la correcta instalación y ejecución del servicio web

Por otra parte, se asume que los requisitos descritos en este documento son estables una vez que se han aprobado por el director del proyecto y el SW esté funcionando en el servidor proporcionado por la UNED y/o la UCM. Por lo que, no se agregarán en primera instancia nuevas funciones a las ya definidas anteriormente. En tal caso, se adjuntará un documento con los cambios precisados, o se procederá a crear una nueva versión de este documento si fuera necesario para reflejar tales cambios.

5.1.2.6. REQUISITOS FUTUROS

Aunque inicialmente se planteó la posibilidad de serializar para anotaciones LAF [11], además de MAF [6] y SynAF [7], las limitaciones de tiempo han dejado esta funcionalidad fuera del alcance del proyecto. Por este motivo, se deja como requisito futuro la posibilidad de ampliar el servicio web actual para que incluya la serialización normalizada LAF.

5.1.3. REQUISITOS ESPECÍFICOS

5.1.3.1. REQUISITOS FUNCIONALES

A continuación se presentan los subsistemas que se han diseñado y que contienen las principales funciones que debe cubrir este proyecto:

1. Subsistema Servicio Web FreeLing (SWF)
2. Subsistema de Configuración y Preparación del Texto (SCPT)
3. Subsistema MAF (SMAF)
4. Subsistema <tiger2/> (S<tiger2/>)
5. Subsistema de Lectura de Etiquetas (SLE)

5.1.3.1.1. Subsistema de servicios web

RF - SWF	**1**
Nombre	**Servicio Web FreeLing**
Descripción	Subsistema principal que se encarga de realizar las llamadas pertinentes que darán lugar a los resultados normalizados del análisis del texto de entrada.
Entrada	Texto.
Salida	Ninguna.
Acción	Llamada al servicio web que realizará el análisis del texto.
Precondición	Tener disponible y en funcionamiento el servidor de aplicaciones.
Poscondición	El servicio web estará funcionando y operativo.

5.1.3.1.2. Subsistema de configuración y preparación del texto

El subsistema descrito en el apartado anterior hace uso del presente subsistema. La configuración y preparación del texto se compone de tres casos de usos, que encapsulan la funcionalidad del mismo, y que se explican a continuación:

RF - SCPT	**2.1**
Nombre	**Opciones de Análisis**
Descripción	Establecerá las opciones disponibles de análisis proporcionadas por la interfaz de *FreeLing*. Es a través de este subsistema donde el usuario elige el tipo de análisis que quiere realizar al texto (morfológico o morfosintáctico).
Entrada	El texto junto con los atributos booleanos correspondientes a cada opción disponible.
Salida	Ninguna.
Acción	Realiza la configuración previa al análisis.
Precondición	Se deben colocar los atributos booleanos según la activación de los módulos que se desee para el análisis. Puede dejarse vacío. En tal caso, se aplicará la configuración por defecto. Además se dará a elegir cuatro opciones: análisis sintáctico (con relaciones de constitución), análisis sintáctico (con relaciones de dependencia), análisis morfosintáctico y análisis morfológico.
Poscondición	Se habrán establecido las opciones para el análisis que se desea. Por defecto se activarán los siguientes: análisis de afijos, reconocimiento de nombres de entidades, detección de multipalabras, números, fechas y cantidades.

RF - SCPT	**2.2**
Nombre	**Tokenizar texto**
Descripción	Se encargará de tokenizar el texto mediante la interfaz de *FreeLing*. El texto de entrada se prepara y segmenta para su análisis.
Entrada	Texto.
Salida	Una lista de palabras tokenizadas.
Acción	Divide el texto en unidades indivisibles preparándolo para su análisis.
Precondición	El texto debe estar en castellano y puntuado correctamente.
Poscondición	La lista de palabras puede estar vacía cuando no haya texto.

RF - SCPT	**2.3**
Nombre	**Splitter del texto**
Descripción	Se encargará de dividir en oraciones el texto de entrada usando la interfaz de *FreeLing*.
Entrada	Lista de tokens.
Salida	Una lista de oraciones.
Acción	Se transforma la lista de entrada en un conjunto de oraciones.
Precondición	Se necesita una lista de palabras previamente tokenizadas.
Poscondición	La lista de oraciones puede estar vacía si no hay nada en la lista de palabras.

5.1.3.1.3. Subsistema de Análisis Morfológico (SMAF)

Este subsistema es el encargado de realizar el análisis morfológico del texto y devolver el documento XML normalizado con ISO/MAF [6]. Para llevar a cabo esta tarea, dicho subsistema se compone de cinco funcionalidades: (1) el propio análisis, (2) el etiquetado *POS Tagging*, (3) la serialización a MAF y, por último (4) y (5) el tratamiento de algunos casos particulares que requieren de un proceso específico en la conversión de la salida de *FreeLing* a su etiqueta en MAF.

RF - SMAF	**3.1**
Nombre	**Análisis Morfológico**
Descripción	Se encargará del análisis morfológico mediante la interfaz de FreeLing.
Entrada	Lista de oraciones.
Salida	Una lista de oraciones analizada morfológicamente.
Acción	Realiza el análisis morfológico de una lista de oraciones de entrada, devolviendo el resultado del análisis realizado.
Precondición	Se necesita una lista de palabras previamente tokenizadas y divididas en oraciones.
Poscondición	Se realiza sobre la misma lista de oraciones pasada como parámetro.

RF - SMAF	**3.2**
Nombre	**Análisis POS Tagger**
Descripción	Se encargará del etiquetado POS (*Part Of Speech*) de la entrada. Se utilizará la interfaz de FreeLing.
Entrada	Lista de oraciones.
Salida	Una lista de oraciones analizada.
Acción	Analiza morfosintácticamente una lista de oraciones, realizando su etiquetado gramatical, que devuelve como salida.
Precondición	Se necesita una lista de palabras previamente tokenizadas y divididas en oraciones.
Poscondición	Se realiza sobre la misma lista de oraciones pasada como parámetro.

RF - SMAF	**3.3**
Nombre	**Etiquetado MAF del Análisis**
Descripción	Se normaliza el análisis morfosintáctico con el etiquetado XML. Se identifican los resultados en <token>, <wordform>, <f>, <fs> y <vAlt>, entre otras etiquetas establecidas por la norma ISO/MAF [6].
Entrada	Lista de oraciones.
Salida	Documento XML, normalizado según la ISO/MAF [6].
Acción	Encapsula en XML una lista de oraciones analizadas morfológicamente.

Precondición	Se necesita una lista de oraciones previamente analizadas morfológicamente.
Poscondición	Se realiza el etiquetado formal según la norma ISO/MAF [6].

RF - SMAF	**3.4**
Nombre	**Adaptación y regla Al-Del**
Descripción	Se adaptan los resultados en el análisis de FreeLing de las contracciones “al” y “del”, adecuándolos a la norma, y retocando el atributo *Form* de cada palabra.
Entrada	Dos palabras contiguas de la oración.
Salida	Se informará si se aplica o no esta regla.
Acción	Comprobar si las dos palabras se usan en la oración de manera abreviada o no.
Precondición	Las palabras están analizadas y no vacías.
Poscondición	Se modifica el atributo *Form* de cada palabra para adecuarlo a la manera que solicita la norma ISO/MAF [6].

RF - SMAF	**3.5**
Nombre	**Adaptación de Acentos y Mayúsculas**
Descripción	Se adecúa la salida del análisis a la norma ISO/MAF [6] introduciendo acentos y/o mayúsculas a palabras que lo necesiten para cumplir con la normalización.
Entrada	Dos palabras (una analizada y otra del texto).
Salida	Palabra normalizada.
Acción	Compara la palabra analizada con la parte del texto en crudo homólogo. Se necesita identificar, comprobar y colocar el acento y/o mayúsculas según la comparación de las mismas.
Precondición	Se necesita una palabra y texto no vacío.
Poscondición	Se actualiza la Forma o "*Form*" de la palabra analizada. La forma de la palabra queda normalizada.

5.1.3.1.4. Subsistema de Análisis Sintáctico (Stiger2)

Este subsistema es el encargado de realizar el análisis morfosintáctico del texto y devolver los documentos XML normalizados con <tiger2/> [9] del análisis de dependencia, o del análisis de relaciones de constitución, o ambos, dependiendo de las opciones de configuración establecidas anteriormente.

Para llevar a cabo esta tarea, dicho subsistema se compone de cuatro funcionalidades: (1) el análisis mediante relaciones de dependencia sintáctica, (2) el análisis mediante relaciones de constitución (*chunker*), (3) la serialización a <tiger2/> de la primera parte del documento (parte `<head>`) y, por último (4) la serialización a <tiger2/> de la segunda parte del documento (parte `<graph>`).

RF – Stiger2	**4.1**
Nombre	**Análisis y visualización del análisis sintáctico (con relaciones de dependencia)**
Descripción	Realizará el análisis sintáctico (*Dependency Parser*) mediante la interfaz de *FreeLing*.
Entrada	Lista de oraciones.
Salida	Resultado *FreeLing* en forma de texto.
Acción	Se realiza un procedimiento recursivo para la muestra de las dependencias de cada palabra en un árbol sintáctico.
Precondición	Se necesita una lista de oraciones previamente analizadas morfológicamente y la solicitud en las opciones de este tipo de análisis.
Poscondición	Se muestra el resultado en forma de texto.

RF - Stiger2	4.2
Nombre	**Análisis y visualización del análisis sintáctico (con relaciones de constitución)**
Descripción	Realizará el análisis sintáctico (*Chart Parser, Chunker*) mediante la interfaz de *FreeLing*.
Entrada	Lista de oraciones.
Salida	Resultado *FreeLing* en forma de texto.
Acción	Se realiza un procedimiento recursivo para la muestra de las dependencias de cada palabra en un árbol sintáctico.
Precondición	Se necesita una lista de oraciones previamente analizadas morfológicamente y la solicitud en las opciones de este tipo de análisis.
Poscondición	Se muestra el resultado en forma de texto.

RF - Stiger2	4.3
Nombre	**Etiquetado <tiger2/> del análisis (parte <head>)**
Descripción	Se identifican los resultados del análisis sintáctico para etiquetarlo con XML, adaptando la parte `<head>` y sus subetiquetas `<meta>`, `<annotations>`, `<features>`, entre otras establecidas por la norma <tiger2/> [9].
Entrada	Resultado del análisis sintáctico y su lista de oraciones.
Salida	Resultado normalizado.
Acción	Encapsula en XML una lista de oraciones analizadas sintácticamente.

Precondición	Se necesita una lista de oraciones previamente analizadas sintácticamente.
Poscondición	Se realiza el etiquetado formal según la norma de <tiger2/>.

RF - Stiger2	**4.4**
Nombre	**Etiquetado <tiger2/> del Análisis (parte <Graph>)**
Descripción	Se identifican los resultados del análisis sintáctico para etiquetarlo con XML, adaptando la parte `<graph>` y sus subetiquetas `<terminals>`, `<t/>`, `</terminals>`,`<nonterminals>`, `<nt>`, `<edge/>`, `</nt>`, `</nonterminals>` entre otras establecidas por la norma <tiger2/> [9].
Entrada	Lista de oraciones.
Salida	Resultado normalizado.
Acción	Encapsula en XML una lista de oraciones analizadas morfológicamente.
Precondición	Se necesita una lista de oraciones previamente analizadas morfológicamente.
Poscondición	Se realiza el etiquetado formal según la norma de <tiger2/>.

5.1.3.1.5. Subsistema de Lectura de Etiquetado (SLE)

RF - SLE	**5**
Nombre	**Lectura de Etiquetado**
Descripción	Realiza la lectura del documento XML que contiene la correspondencia de las etiquetas utilizadas por la aplicación *FreeLing* a un comentario comprensible por cualquiera.
Entrada	Etiquetas usadas por *FreeLing*: *EAGLES, Syn* y *Func* y contenido de la misma. Ejemplo: *EAGLES*, "NCMS01".
Salida	Significado o comentario acerca de la etiqueta. Ejemplo: "Nombre: Común, Masculino, Singular".
Acción	Leerá una etiqueta de un archivo XML devolviendo su comentario asociado. Formato del documento: `<tag>"etiquetax"</tag> <comment>"comentariox"</comment>`
Precondición	La etiqueta y el tipo (EAGLES, Syn, Func) existen.
Poscondición	Se cerrarán los archivos XML leídos.

5.1.3.2. REQUISITOS DE INFORMACIÓN DEL SISTEMA

RIS-01	Texto de entrada
Descripción	El sistema manejará la información introducida con las reglas ortográficas y gramaticales del español como base.

RIS-02	Establecimiento de comunicación
Descripción	El servidor deberá contar con una comunicación a la red HTTP necesaria para la transmisión de datos con el servicio web.

RIS-03	Disponibilidad
Descripción	El servicio web estará disponible para su uso las 24h del día.

5.1.3.3. REQUISITOS GENERALES DEL SISTEMA

RS-01	Estabilidad
Descripción	El servidor deberá permanecer íntegro y estable durante un largo periodo de tiempo puesto que estará a disposición de gran número de clientes.

RS-02	Almacenaje
Descripción	El servicio web no guardará ningún tipo de registro del cliente, ni nada relacionado con su condición o motivo de la utilización del sistema

RS-03	Continuidad energética.
Descripción	El servidor deberá estar funcionando sin interrupciones.

RNF-01	Requisitos de interfaz
Descripción	La solicitud se realizará mediante el protocolo WSDL por parte de cualquier cliente que solicite la llamada el servicio web.

RNF-02	Accesibilidad
Descripción	El cliente que acceda al servicio web puede ser de cualquier tipo, independiente del lenguaje de programación y la plataforma tecnológica que provenga, sin discriminación alguna.

RNF-03	Escalabilidad
Descripción	En previsión de futuras necesidades, el sistema permitirá escalabilidad horizontal y vertical.

5.1.3.4. REQUISITOS DE RENDIMIENTO

RR-01	Requisito Rendimiento
Tiempo máximo	Tiempo de respuesta máximo: 60 segundos.
Peticiones por segundo	10 peticiones por segundo.
Nº usuarios concurrentes	5 usuarios concurrentes.
Rendimiento general	Tiempo de respuesta medio: 30 segundos.

Comentarios	El número de peticiones por segundo puede variar en función del número de clientes que accedan al servicio en ese momento. Como mucho se servirán 10, el número máximo de clientes en el sistema, permitiendo 5 usuarios concurrentes.

5.1.3.5. REQUISITOS DE DESARROLLO

El ciclo de vida para desarrollar el producto será realizado bajo la metodología en cascada, debido a que es un desarrollo pequeño para la realización de un servicio web y esta metodología es la más utilizada en estos casos. El desarrollo contará con la participación de tres personas, las integrantes del grupo, que formarán parte de todo el proceso de desarrollo.

5.1.3.6. REQUISITOS TECNOLÓGICOS

Los requisitos mínimos de la aplicación serán los siguientes:

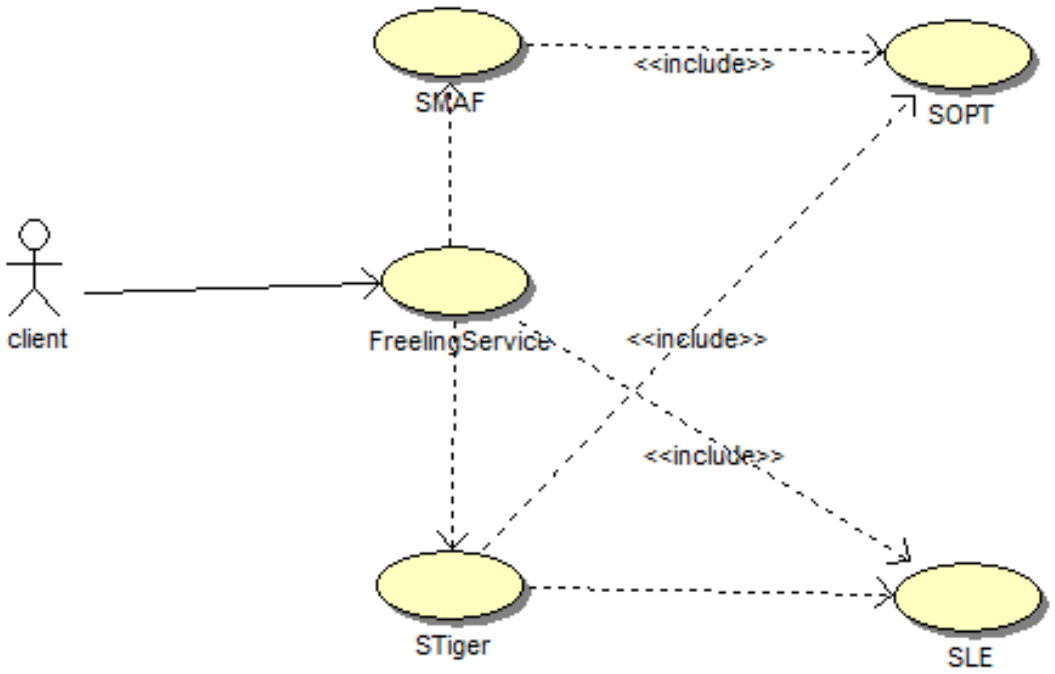
- Procesador de 32 bits/64 bits a 2 gigahercio (GHz) o más.
- Memoria RAM de 1 gigabyte (GB) (32 bits) o 2 gigabyte (64 bits).
- Espacio disponible en disco rígido de 16 GB (32 bits) o 20 GB (64 bits).
- Dispositivo gráfico DirectX 9 con controlador WDDM 1.0 o superior.
- Tarjeta de Red.

Dichos requisitos son similares a los de Linux Ubuntu, que será el sistema operativo para el cual se creará se establecerá la aplicación.

5.2. ANÁLISIS Y DISEÑO

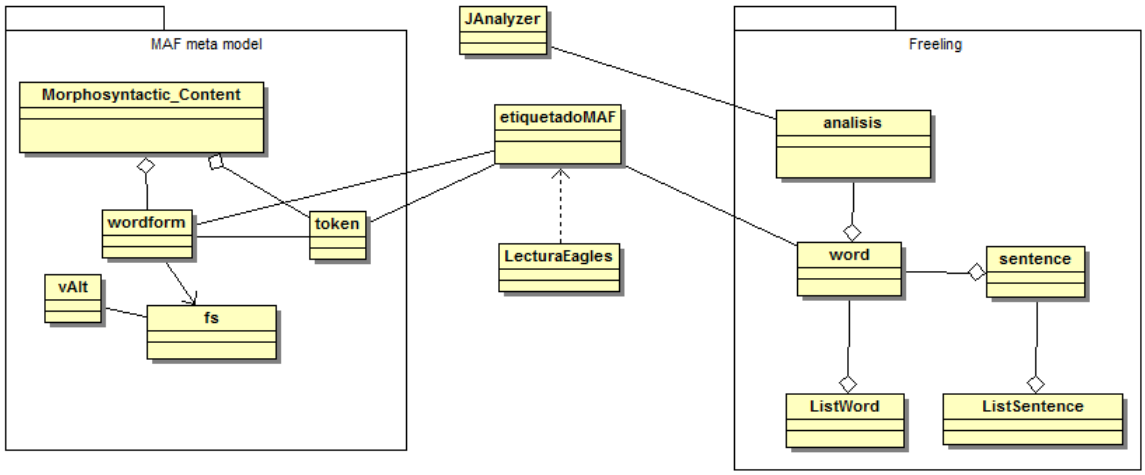
En este apartado se realizará el análisis y diseño de la aplicación a entregar teniendo en cuenta los requisitos ERS para la realización de este proyecto. Se dispondrán a continuación los diagramas necesarios para establecer el "cómo" desarrollar este servicio web explicando mediante diagramas UML el sistema.

Se identifican los principales sub-módulos esbozados en el documento de requisitos presentando el siguiente diagrama:



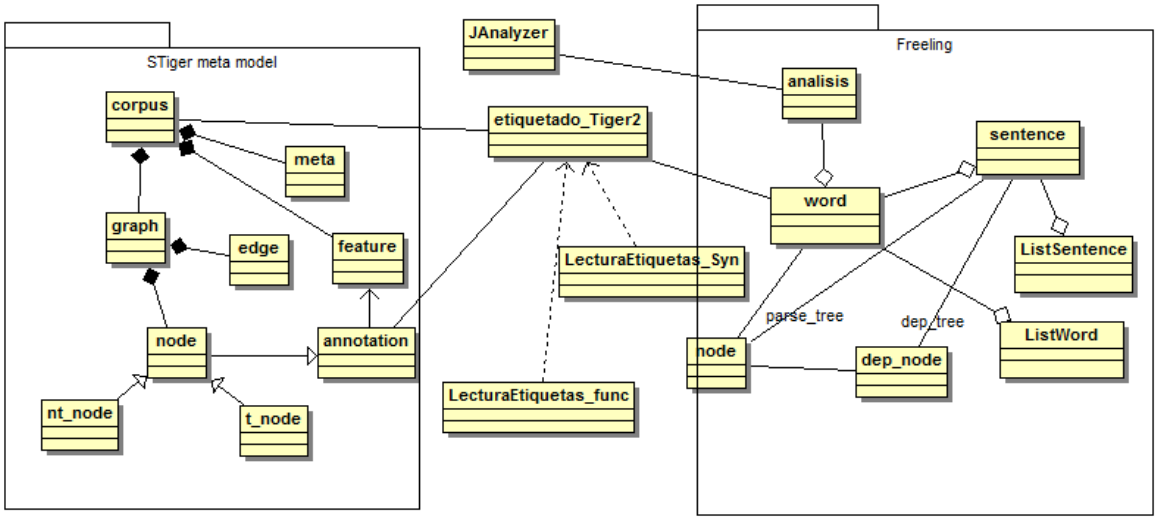
Tanto el módulo SMAF como SSynAF necesitarán de SOPT para capturar las opciones y personalización del análisis por parte del cliente sobre su input (SOPT) además de la correcta interpretación del etiquetado de *FreeLing* para ayudar al proceso de normalización de la solicitud (SLE).

A continuación se refleja en un diagrama de clases cómo interactúan las entidades proporcionadas de FreeLing, tomando como referencia el documento del profesor Lluís Padró[18]. Así mismo partiremos de la referencia del diagrama plasmado en el documento de normalización [6] y enlazando estos dos ámbitos elaboramos el siguiente diseño para el módulo SMAF correspondiente a su normalización:



[18] FreeLing: documento descriptivo del modelado. http://nlp.lsi.upc.edu/papers/padro11.pdf. Consultado en octubre de 2012.

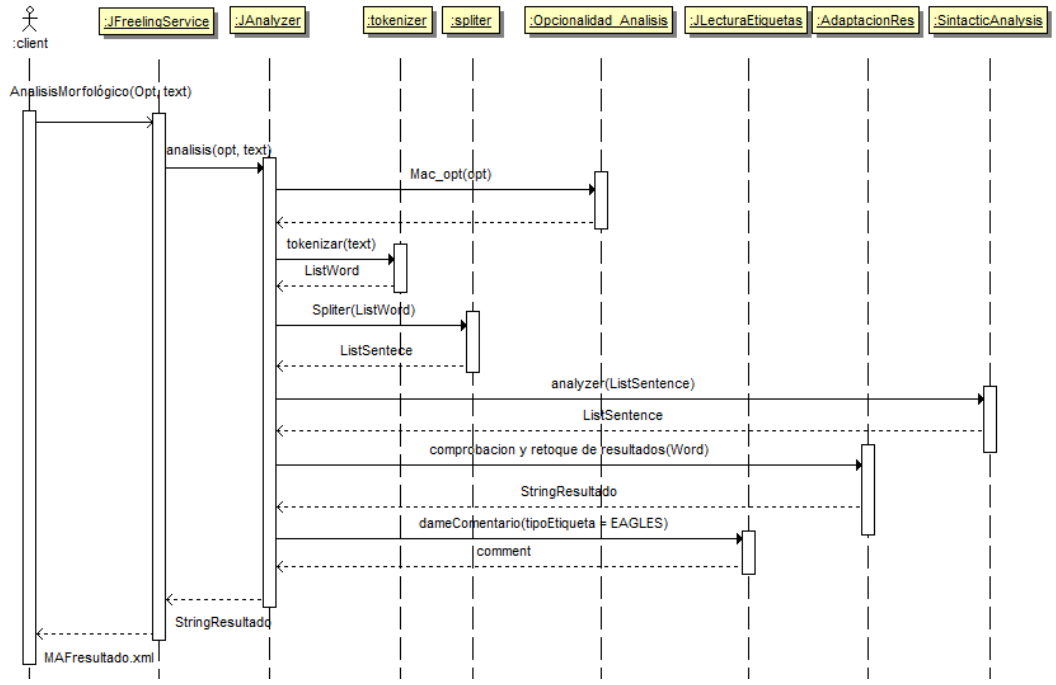
Así mismo el diseño del sistema SSynAF tomando como referencia el estándar de SynAF2 es el siguiente [9]:



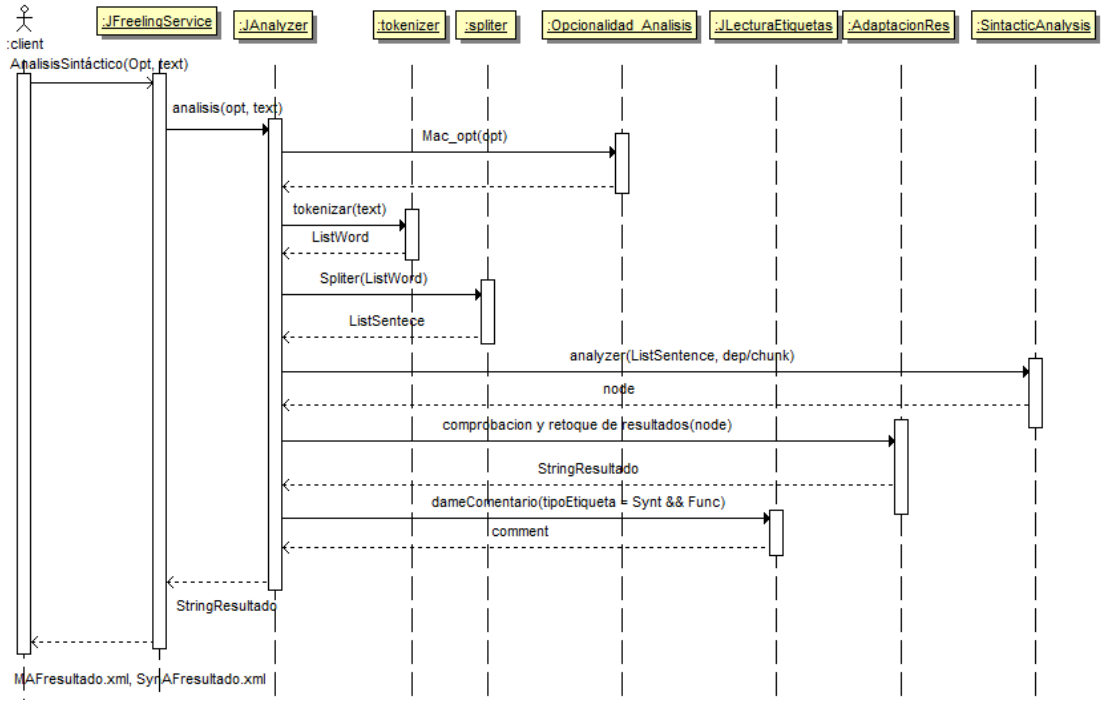
Tanto SynAF meta modelo como MAF meta modelo se corresponden con el conjunto de clases que los estándares definen para la consecución de la normalización en XML. El puente de unión presentado en los diagramas hará uso de los paquetes tanto de *SynAF/MAF* como del paquete *FreeLing*. Una vez identificadas las etiquetas se procederá a encapsular cada una de ellas con la información que FreeLing nos proporciona, para ello se hará uso de sus clases y solo de las necesarias para el análisis morfológico y sintáctico.

La manera de interactuar cada entidad en orden de llamada se describe en los siguientes diagramas de secuencia:

SSMAF:



SSynAF:



5.3. IMPLEMENTACIÓN

5.3.1. INSTALACIÓN DE LAS LIBRERÍAS FREELING

A continuación, expondremos cómo se inició el desarrollo del proyecto, atendiendo a las vicisitudes y problemas surgidos, sobre la marcha del mismo. Esta sección, además, sirve como una guía esclarecedora de los errores más comunes para la realización de proyectos similares, y ofrece información para encarar de forma idónea la fase de arranque.

Los recursos iniciales se proporcionan y referencian en la página web de *FreeLing*[19]. Desde aquí se dispone tanto del software necesario a descargar, así como, su documentación, más un foro de ayuda en el que colaboran distintos desarrolladores, miembros del equipo *FreeLing,* además de los propios usuarios registrados.

Las primeras cuestiones planteadas al querer poner en marcha un servicio web con el paquete proporcionado por la librería son:

- ¿En qué plataforma se debe realizar el proyecto?
- ¿Qué herramientas IDE se necesitan para lograr un servicio web?
- ¿Cómo se instala *"FreeLing"* y sus librerías?
- ¿Cómo se usan las librerías de *"FreeLing"* en el servicio web?
- ¿Existen incompatibilidades?

En los siguientes apartados se exponen las decisiones que se fueron tomando relativas a las cuestiones mencionadas, la consecución de problemas que fueron surgieron y las medidas adoptadas hasta conseguir hacer funcionar *FreeLing* en un servicio web.

De todo el proyecto, la parte más ardua y costosa ha sido, sin duda, la puesta en marcha. Lo que inicialmente se pensaba que era trabajo de un par de días se convirtió en una escalada de dificultades que se tardó tres meses en resolver.

[19] FreeLing. http://nlp.lsi.upc.edu/FreeLing/. Consultado en octubre de 2012.

5.3.1.1. Problemas con Windows

Debido a que *FreeLing* disponía en su página de descargas, como una de las primeras opciones, el paquete compilado para Windows y, el conocimiento sobre las herramientas de desarrollo de este sistema operativo eran muy familiares para el grupo, se decidió usar la plataforma Windows 7 de 64 bits con un potente IDE, como Visual Studio 2010.

El código del paquete está escrito en C++, por lo que, lo único que se debía hacer era crear un servicio web en C++ al que incorporar la librería de *FreeLing*. En este punto nos encontramos con el primer problema, la plantilla predetermina que te permite crear servicios web en C++ para Visual 2010 no existe, el lenguaje con el que posees documentación y soporte para servicios web es C#.

Debido a la falta de información sobre la creación de servicios web desde cero con éste lenguaje, no se pudo avanzar es esta línea de trabajo, la cual, se descartó por la cantidad de tiempo que se había consumido en esta alternativa.

Se buscó entonces una plantilla para servicios web en C++, que se podía usar con versiones anterior de Visual Studio. Se obtuvo del apartado de descargas de la página web de *FreeLing* el denominado "*FreeLing-3.0-win-fix.zip*"[20] que contiene las librerías compiladas para Windows y, también se descargó el proyecto original con el código fuente nativo "*FreeLing-3.0.tar.gz*"[21] para instalarlo en el directorio raíz.

Una vez descargados estos dos paquetes, se siguieron las instrucciones del documento "*Readme.txt*" de "*FreeLing-3.0-win-fix.zip*"[22] para desplegar las librerías en nuestro

[20] FreeLing: sección de descarga de paquetes. http://devel.cpl.upc.edu/FreeLing/downloads?order=time&desc=1. Consultado en octubre de 2012.

[21] Véase el comentario a la nota al pie anterior.

[22] Véase el comentario a la nota al pie anterior.

sistema operativo. Además, para compilar *FreeLing* en Windows, paso indispensable en la instalación, se siguieron las instrucciones del "*Readme.txt*" que está en el paquete "*FreeLing-3.0.tar.gz*" en la ruta "*FreeLing-3.0**msv".*

Se consiguió llevar a cabo la integración pero, el servicio web no arrancaba, después de un tiempo buscando información al respecto y leyendo en distintos foros, llegamos a la conclusión que la plantilla se había desechado en versiones posteriores porque no funciona bien.

Entonces, se decidió cambiar el servicio web para programarlo en C#. No obstante, debido a que las librerías de *FreeLing* están escritas en C++, el uso de sus funcionalidades es incompatible con C#, pues no es un componente COM (Component Object Model). De hecho, se producía un error al añadir la librería *FreeLing.dll* a nuestro servicio web montado en C#. Esta DLL es la que se proporciona en el paquete "*FreeLing-3.0-win-fix*" [23] de la ruta "*FreeLing_win**FreeLing**lib*" que contiene la librería compilada para Windows.

La solución inmediata a este problema fue la creación de las llamadas clases envoltorio (*wrappers*) con la tecnología *PInvoke* (*Platform Invocation Services*)[24], mediante el cual todas las funciones de la librería que posee "*FreeLing*" serían envueltas y retocadas para poder ser utilizadas desde C#.

La cantidad de funcionalidad del paquete que se debía envolver, que suponía, una recodificación de muchas líneas de código, unido al desconocimiento a nivel de programación que se disponía del paquete, hizo descartar esta posible alternativa.

La elección de esta solución era claramente inviable en el marco temporal al que debemos someternos en el proyecto. Esto es así, debido a la cantidad de archivos y

[23] Véase el comentario a la nota al pie anterior.

[24] PInvoke. http://www.pinvoke.net/. Consultado en octubre de 2012.

clases que todo el entorno *FreeLing* contiene en su código fuente (alrededor de 200 MB). Un esfuerzo con, además, alta probabilidad de aparición de errores.

Esta propuesta sigue una filosofía totalmente contraria al paradigma de abstracción, dónde el uso de los esquemas de clases o funciones ya dadas no debe de romper la barrera de la reimplementación[25].

Después de esto, sólo quedaba preguntar en el foro de *FreeLing* para ver si podían ayudarnos. Su respuesta fue la siguiente:

> *"El equipo de desarrollo de FreeLing no proporciona soporte para Windows"* [26]

Con esta respuesta y después de todos los problemas encontrados, no era viable seguir intentando usar *FreeLing* en Windows. En ese momento se decidió cambiar de sistema operativo.

5.3.1.2. Problemas con Linux

Dado que el soporte y experiencia de los desarrolladores de *FreeLing* es principalmente en Linux, y varias de sus distribuciones, se decidió iniciar de nuevo el proyecto para Ubuntu *12.04 precise* de *64 bits.* Y debido a que con C++ se habían encontrado también muchos problemas, se decidió usar la API de Java que tiene *FreeLing* para Linux.

Tras realizar la instalación completa de las librerías, para su posterior compilación, se generaron errores de enlace con librerías internas. Consultando de nuevo a los desarrolladores de *FreeLing* se obtuvo la siguiente contestación:

---

[25] Wikipedia: Programacion Orientada a Objetos. http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos. Consultado en marzo de 2013.

[26] FreeLing: foro. http://nlp.lsi.upc.edu/FreeLing/index.php?option=com_simpleboard&Itemid=65. Consultado en marzo de 2013.

"This linker error is usually related to 32/64 bits architecture mismatch between the compiled code (the library in this case) and the linked library (-lFreeLing). It may be that FreeLing is compiled in 32 bits and you are working on a 64bits system"[27].

El problema era que las librerías eran incompatibles con el sistema operativo a 64 bits. Se optó entonces por una máquina virtual de *Ubuntu de 32 bits* utilizada en el software *VMware* y se realizó de nuevo la instalación y compilación de las librerías. Aun siguiendo paso a paso las instrucciones dadas, tuvimos que realizar el proceso cuatro o cinco veces desde el inicio para la creación del ".JAR", que es generado tras la compilación de "*FreeLing*". El paquete que es necesario instalar para su ejecución en *Ubuntu 12.04 precise* es "*FreeLing-3.0-precise.deb*"[28]. Para la instalación se siguió paso a paso el manual. Existen varios modos de instalación: desde "*freling-3.0-precise.deb"* descargado o desde el paquete fuente, principalmente. El código nativo es el originario del proyecto, que permite la instalación desde cero para cualquier sistema operativo, aunque el nivel de complejidad es alto. Por otro lado, el fichero DEB te ofrece el paquete ya compilado para Linux. En este caso, se optó por el fichero DEB, dado que es la manera más fiable de que se instale en el sistema.

La utilización de Java implica seguir las instrucciones soportadas en la documentación de *FreeLing* para poder generar el paquete *JAR* y, por ende, todos los archivos Java necesarios para el nuevo entorno. Estas instrucciones se encuentran dentro del paquete "FreeLing-3.0.tar.gz" en la ruta "\FreeLing-3.0\APIs\java\readme".

[27] FreeLing: foro. http://nlp.lsi.upc.edu/FreeLing/index.php?option=com_simpleboard&Itemid=65. Consultado en octubre de 2012.

[28] FreeLing: sección de descargas. http://devel.cpl.upc.edu/FreeLing/downloads?order=time&desc=1. Consultado en octubre de 2012.

Al ejecutar el archivo "*makefile"* se crea el paquete "*.JAR*", así como, los archivos "*.so*" que se deberán colocar manualmente dentro del directorio "/usr/lib" para evitar problemas con la variable de entorno *LD_LIBRARY_PATH*. En otro caso, la variable de entono *LD_LIBRARY_PATH* no se inicializará correctamente y esto dará problemas en el enlazado y compilación del código.

Es imprescindible tener muy presente, antes de comenzar la compilación de la librería, todos los paquetes, y las versiones de los paquetes, que necesita tener instalado el sistema para que no se produzca ningún error de dependencia en el "*makefile*" que crea el "*.JAR". Además, se deben incluir con sumo cuidado todas las variables de entorno necesarias.

Una vez llevado a cabo todo el proceso, generado el archivo Java e integrado en el servicio web bajo la plataforma de desarrollo Netbeans, se tomó como ejemplo el archivo "*analyzer.java*" ofrecido en las librerías.

Todavía quedaba solucionar un último error antes de obtener una ejecución final. Cuando se compilaba el código de ejemplo, ya bien integrado con la librería "*FreeLing*", se obtenía un error de enlazado que no provenía de ningún tipo de fallo provocado por una mala instalación o compilación.

Este problema también se consultó en el foro de *FreeLing* pero únicamente remitieron como solución la lectura del manual.

Se descubrió, finalmente, mediante el mecanismo de prueba y error, que era necesario incluir "*FreeLing_javaAPI.so*" generado por el "*makefile*", en la ruta "*/usr/lib*"[29].

[29] FreeLing: foro. http://nlp.lsi.upc.edu/FreeLing/index.php?option=com_simpleboard&Itemid=65&func=view&id=3073&catid=3. Consultado en marzo de 2013.

Y de esta forma, se logró compilar y ejecutar el ejemplo del analizador que ofrecen las librerías. Esta vez el éxito fue claro, se había completado la primera fase de la puesta a punto. Falta ahora, utilizar el paquete integrado en el servicio web para proceder a la normalización de las anotaciones proporcionadas por *FreeLing*.

5.3.2. CODIFICACIÓN DEL PROYECTO

Teniendo en cuenta los diagramas vistos podemos realizar la codificación del proyecto sirviéndose de guía lo reflejado en ellos. El desarrollo se realiza en el entorno o *IDE Netbeans 7.2.1* bajo la plataforma *Linux 12.04 precise* usando la librería *FreeLing.jar* generada en la fase previa.

También se creará una aplicación web que se usará como ejemplo de un cliente para realizar las pruebas del *web service* que se montará y al que se nombrará *FreeLingService*. El cliente de ejemplo será creado en este mismo IDE dándole la UDDI del servicio y de esta manera probar la funcionalidad del servicio.

En este apartado se describirán parte del código implementado por la aplicación realizando una traza de ejecución para los módulos principales explicando su recorrido desde la entrada de texto hasta la generación de los archivos xml.
A continuación se desplegará el recorrido de ejecución del método principal AnalysisService.

5.3.2.1. CAPTURA DE DATOS DEL CLIENTE Y LLAMADA A LA FUNCIÓN NORMALIZADORA

Dentro del archivo *JFreeLingService* el cual se corresponde con la parte del servicio del proyecto esta implementado con el método *AnalysyService()* por el cual recibirá por parámetro todas las opciones de análisis que recibe al ser llamado desde un cliente En la primera parte se prepara las variables *string* de los nombres a los archivos que se crearán en ejecución además del *string* que contiene los enlaces a la descarga de los archivos xml generados.

Se procede a crear *JAnalyzer* y se da el texto por parámetro asi como las opciones recibidas del servicio. Una vez realizado los resultados del análisis y normalización de éste, se creará un archivo xml volcando los resultados en él. Esta parte está debidamente capturada con la sección try catch.
Para finalizar el método del *web Service* devolverá un *string* que tendrá sintaxis xml y se corresponderá a la sección `<href>` para el enlace de descarga del archivo generado. Esta variable puede poseer varios enlaces a los diferentes archivos generados en función del tipo de análisis que se pide y que está reflejado en las opciones.

```
@WebMethod(operationName = "AnalysisService")
    public String AnalysisService (String frase, JOpt opciones){

     String nombreMAF = "";
     String nombreMAFParser = "";
     String nombreMAFDep = "";
     String nombreTigerParser = "";
     String nombreTigerDep = "";
     String nombreFreeling = "";

     nombreMAF =
"/home/guille/NetBeansProjects/FreelingWebAplication/build/web/spanish.example.maf.xml";
     nombreMAFParser =
"/home/guille/NetBeansProjects/FreelingWebAplication/build/web/spanish.example_MAF_parse
r.maf.xml";
     nombreMAFDep =
"/home/guille/NetBeansProjects/FreelingWebAplication/build/web/spanish.example_MAF_dep.m
af.xml";
     nombreTigerParser =
"/home/guille/NetBeansProjects/FreelingWebAplication/build/web/spanish.example_tiger_par
ser.standoff.tiger2.xml";
     nombreTigerDep =
"/home/guille/NetBeansProjects/FreelingWebAplication/build/web/spanish.example_tiger_dep
.standoff.tiger2.xml";
     nombreFreeling  =
"/home/guille/NetBeansProjects/FreelingWebAplication/build/web/SalidaFreeLing.txt";
```

```
      String xmlMAF = "";
      String xmlTigerParser = "";
      String xmlTigerDep = "";

         // Donde alojaremos los enlaces para descarga de los archivos en sintaxis HTML
      String enlace ="";


         JAnalyzer a = new JAnalyzer();

         try{
             String res = a.dameResultados (frase, opciones);

             if(opciones.morphologicalAnalysis==true || opciones.POStagAnalysis==true){
                 xmlMAF = a.dameXml_MAF("maf");
                 hazArchivo(xmlMAF, nombreMAF);
                 enlace += "<br><a
href=\"http://localhost:8080/FreelingWebAplication/spanish.example.maf.xml\"> exampleMAF
</a><br> ";
             }

             // Parser
             if(ChunkAnalysis==true){
                 xmlMAF = a.dameXml_MAF("parser");
                 hazArchivo(xmlMAF, nombreMAFParser);
                 xmlTigerParser = a.dameXml_Tiger("parser");
                 hazArchivo(xmlTigerParser, nombreTigerParser);

                 enlace += "<br><a
href=\"http://localhost:8080/FreelingWebAplication/spanish.example_MAF_parser.maf.xml\">
exampleParserMAF </a> ";
                 enlace += "<br><a
href=\"http://localhost:8080/FreelingWebAplication/spanish.example_tiger_parser.standoff
.tiger2.xml\"> exampleTigerParser </a> ";
             }

             // Dep
             if(DepAnalysis==true){

                 xmlMAF = a.dameXml_MAF("dep");
                 hazArchivo(xmlMAF, nombreMAFDep);
                 xmlTigerDep = a.dameXml_Tiger("dep");
                 hazArchivo(xmlTigerDep, nombreTigerDep);

                 enlace += "<br><a
href=\"http://localhost:8080/FreelingWebAplication/spanish.example_MAF_dep.maf.xml\">
exampleDepMAF </a>";
                 enlace += "<br><a
href=\"http://localhost:8080/FreelingWebAplication/spanish.example_tiger_dep.standoff.ti
ger2.xml\"> exampleTigerDep </a>";
             }

             //}
             if(freeling=true){
                 hazArchivo(res, nombreFreeling);
                 enlace += "<br><a
href=\"http://localhost:8080/FreelingWebAplication/SalidaFreeLing.txt\"> exampleFreeLing
</a> ";
             }

         }
         catch (Exception e) {
             e.printStackTrace();
         }
      return enlace;
```

5.3.2.2. ESTABLECIMIENTO DE LAS OPCIONES CAPTURADAS Y NORMALIZACIÓN DE FREELING.

Desde *JFreeLingService* se llama al método de análisis de la clase *JAnalyzer* y este se encargara de:

En la primera parte carga la librería de FreeLing necesaria para usar su interfaz. A continuación establece las opciones para el análisis. Algunas opciones internas las se establecen por defecto. Existen 11 atributos boleanos + 4 de aspecto de análisis, es decir, Morfológico, *POS Tagging*, Parser y Dep.

Se realiza la activación de los módulos de FreeLing necesarios para el análisis así como archivos de configuración que se tomarán en cuenta para la generación de la salida.

Con el texto capturado por parámetro de la clase llamadora se realizará la *tokenización* de este texto en crudo devolviendo una lista de palabras o *ListWord* y a su vez se usará esta lista para dividirlas con *Spliter* dando lugar una lista de oraciones *ListSentence*.

Una vez hecho esto se procede al tipo de análisis pedido por parámetro en la variable *op*.

A continuación se realiza la normalización del resultado del análisis en la lista de oraciones, es decir mediante los métodos encargados de recubrir en XML los resultados del análisis. **MAF** se usará para recubrir tanto el análisis Morfológico como el POSTag y **<tiger2/>** para el Parser y Dep.

Para finalizar se generará los archivos XML físicos en el servidor con el resultado normalizado además del resultado en crudo de FreeLing por consola.

```
public String dameResultados(String text, JOpt op) throws IOException{

System.loadLibrary( "freeling_javaAPI" );
Util.initLocale( "default" );


// Modulo de opciones de FreeLing
MacoOptions op = new MacoOptions( LANG );
```

```
    // op.setActiveModules(false, true, true, true, true, true, true, true, true, true,
false );
    op.setActiveModules(
      false, Op.AffixAnalysis,Op.MultiwordsDetection, Op.NumbersDetection, true,
      Op.DatesDetection, Op.QuantitiesDetection, true, true, Op.NERecognition, false );

    op.setDataFiles(
      "",
      DATA + LANG + "/locucions.dat",
      DATA + LANG + "/quantities.dat",
      DATA + LANG + "/afixos.dat",
      DATA + LANG + "/probabilitats.dat",
      DATA + LANG + "/dicc.src",
      DATA + LANG + "/np.dat",
      DATA + "common/punct.dat",
      DATA + LANG + "/corrector/corrector.dat" );

    // Creacion de analizadores necesarios.
    Tokenizer tk = new Tokenizer( DATA + LANG + "/tokenizer.dat" );
    Splitter sp = new Splitter( DATA + LANG + "/splitter.dat" );
    Maco mf = new Maco( op );
    String res = "";

    HmmTagger tg = new HmmTagger( LANG, DATA + LANG + "/tagger.dat", true, 2 );
    ChartParser parser = new ChartParser(
      DATA + LANG + "/chunker/grammar-chunk.dat" );
    DepTxala dep = new DepTxala( DATA + LANG + "/dep/dependences.dat",
      parser.getStartSymbol() );
    Nec neclass = new Nec( DATA + LANG + "/nec/nec-ab.dat" );

    UkbWrap dis = new UkbWrap( DATA + LANG + "/ukb.dat" );

      String line = text;
      texto = text;

      // Extract the tokens from the line of text.
      ListWord l = tk.tokenize( line );

      // Split the tokens into distinct sentences.
      ListSentence ls = sp.split( l, false );

      // Perform morphological analysis
      mf.analyze( ls );

      // Perform part-of-speech tagging.
      if (POStagAnalysis==true)
        tg.analyze( ls );

      dis.analyze( ls );

      if (morphologicalAnalysis==true || POStagAnalysis==true){
        abrirMAF("tagged");
        res = dameResultados( ls, "tagged");
        cerrarMAF("tagged");
      }
        // Perform named entity (NE) classificiation.
               neclass.analyze( ls );

        // Chunk parser
        if (ChunkAnalysis==true){


            prepararHeadSynAF();
            abrirBodySynAF();
            abrirMAF("parser");

            parser.analyze( ls );
            res += dameResultados( ls, "parsed");

            cerrarMAF("parser");
            cerrarBodySynAF("parser");
        }
```

```
        // Dependency parser
        if(DepAnalysis){
            texto = text; // Necesitamos de nuevo el texto en crudo para analizar, con
el analisis parser se borró.
            prepararHeadSynAF();
            abrirBodySynAF();
            abrirMAF("dep");


            dep.analyze( ls );
            res += dameResultados( ls, "dep" );
            cerrarMAF("dep");
            cerrarBodySynAF("dep");
        }
       return res;
```

5.3.2.3. MAPEO DE LOS RESULTADOS DE FREELING Y RESOLUCIÓN DE CONFLICTOS CON LA NORMALIZACIÓN.

Para finalizar se muestra la sección del código encargada de gestionar los resultados de FreeLing y darle formato con las etiquetas del estándar. Para ello lo que se hace es:

Según sea el análisis pedido se enviará la traza de ejecución al tipo de análisis realizado, tomaremos como ejemplo la traza hacia el análisis Morfológico. Cabe mencionar que un análisis Parser o Dep generaran dos XML uno correspondiente al recubrimiento **<tiger2/>** y otro a su referencia **MAF** necesaria.

Para empezar se extrae de la lista de oraciones *ListSentence* que corresponde al texto analizado, su lista de palabras *ListWord,* y por cada oración a su vez se extrae el *Form* llamando al método *getForm()* de cada *Word* así como su etiqueta EAGLE necesaria con el método *getTag()* para interpretar su comentario asociado. A continuación se esclarece si cada palabra contiene elementos retokenizables, es decir, si tiene subpalabras que deban ser debidamente etiquetadas (EJ: dí-se-lo) con sus correspondientes `<token>`. A su vez se colocara la etiqueta `<wordform>`.

Se comprobará si existen casos de abreviaturas *"Al"* o "*del*", acentos, mayúsculas y todas las posibles incompatibilidades de la salida de FreeLing con la que pida el estándar. La nomenclatura "*join Left*" estará establecida también en el resultado normalizado. Se gestiona estos casos para la correcta salida de resultados.

Cada etiqueta tendrá un comentario asociado que será su significado. Para conseguir este comentario se crea una clase *JLeerXML*. Mediante esta clase se llamará al método *damecomentario()* con el *tag* de la palabra por parámetro y este se encargará de leer un archivo de configuración en XML *config.xml* en la cual se establecen todas las etiquetas EAGLES posibles. Se extraerá el comentario asociado al *tag* recibido por parámetro desde este archivo *config.xml* y lo devolverá para introducirlo en el *string* xmlMAF resultado en la sección correspondiente a explicar el sentido de las etiquetas de cada *Wordform*.

Se irá acumulando estos resultados etiquetados en el string *xmlMAF* por cada palabra gestionada hasta acabar de normalizar todas las oraciones de la lista "*ListSentence*" y devolviendolo en la variable *xmlMAF*.

```
private static String dameResultados( ListSentence ls, String format) {
         String res = "";
      inicializarVariables();
      if( format == "parsed" ) {
            res +=  "<!-------- CHUNKER results ---------->\n";

            for( int i = 0; i < ls.size(); i++ ) { // Para cada frase
              TreeNode tree = ls.get( i ).getParseTree();

              //xml="";
              idSentence=i+1;
              abrirSentenceSynaf();
              res += printParseTree( 0, tree );
              cerrarSentenceSynaf();
            }

            xmlTigerParser = xmlTiger;
            xmlMAFParser += xml;
            //xmlMAFParser = xmlMAF;
        }
        else if( format == "dep" ) {

          res +=  "-------- DEPENDENCY PARSER results ----------\n" ;
          for( int i = 0; i < ls.size(); i++ ) {
            TreeDepnode deptree = ls.get( i ).getDepTree();
            xml_terminalsPrincipal ="";
            idSentence=i+1;

            // Necesitamos una lista en la que agregaremos los nodos a procesar,
            // Se agregan de manera desordenada debido al analisis dep
            listaNodosProcesados = new ArrayList<tratamientoPalabra>();

            abrirSentenceSynaf();
            res += printDepTree( 0, deptree );

            // Ordenamos la lista de nodos a procesar por el id de cada nodo
            Collections.sort(listaNodosProcesados);

            int count;
            // Extraemos los nodos a procesar en orden y lo procesamos normalizandolos
            for(int m=0; m<listaNodosProcesados.size(); m++){
                tratamientoPalabra t = (tratamientoPalabra) listaNodosProcesados.get(m);
                count = t.getID();
                normalizarWord_MAF(null, t.getDepnode(), "dep");
```

```
            }
            cerrarSentenceSynaf();
          }

          // Agrupamos las partes al xml de salida
          xmlMAFDep += xmlTokens + "\n\n" + xmlWordForms;
          xmlTigerDep = xmlTiger;
        }
        else{

            res += "-------- TAGGER results -----------\n";

            idToken = 1;
            idWordForm = 1;
            idSentence = 1;
            idTerminal = 1;

            // get the analyzed words out of ls.
            for( int i = 0; i < ls.size(); i++ ) { // For sentence

                                Sentence s = ls.get( i );
                esAlDel=false;
                for( int j = 0; j < s.size(); j++ ) { // For words

                    // Inicializacion de las variables palabra que vamos a usar
                    w1=null;
                    w2=null;
                    subw=null;
                    k=0;
                    retokenizable = false;
                    multiWord = false;
                    w1 = s.get( j );
                    String palabra = w1.getForm();
                    retokenizable = w1.hasRetokenizable();
                    multiWord = w1.isMultiword() && w1.getNWordsMw()>1;
                    if (retokenizable || multiWord){  // Retokenizable: Dí-se-lo;
Multiword: Mariano-Rajoy

                        if (retokenizable)
                            // Descompone y trata los subtokens contenido en la
palabra. Ej: Díselo = Dí-se-lo
                            tratarRetokenizable("tagger");
                        else
                            // Descompone y trata los subwords contenido en la
palabra. Ej: Marriano_Rajoy = Mariano-Rajoy
                            tratarMultiWord("tagger");
                    }else{
                           // La palabra no contiene ni subtokens ni subpalabras.
                           if (j+1 < s.size()){ // No nos salimos del total de palabras
                               w2 = s.get( j+1 );  // Me sirve para comprobar la regla
de+el o a+el despues
                           }else
                               w2=null;
                           tratarPalabra("tagger");
                    }

                } // end For word

            } // end For sentences
            xmlMAF += xml;
        } // end else Tager results

      return res;

}
```

5.4. PRUEBAS

Las pruebas realizadas al servicio web se han llevado a cabo a través de una clase explícita programada para ello y con un cliente web que llama al servicio.

5.4.1. PRUEBAS FUNCIONALES

Realizadas mediante la clasePrAnalyzer.java que contiene un método principal con un parámetro de entrada que recoge el texto a analizar. Las pruebas han permitido detectar y solucionar los siguientes problemas:

1. Incompatibilidad acentos y mayúsculas.

Uno de los problemas encontrados en el proceso de normalización de la salida de *FreeLing* es tratar los acentos y mayúsculas. *FreeLing* proporciona un método en cada palabra o *Word* que da la forma o *form* de dicha palabra mediante el método *getForm()*. Por ejemplo, para la palabra *Díselo* se necesita descomponer en 3 subtokens *Dí-se-lo* pero al tratar de descomponer y recolocar en su correspondiente etiqueta wordform, la salida evita el acento del primer subtoken, esto es, muestra *Di* en vez de *Dí*. Si recurrimos a su segunda forma, da la palabra con acento pero sin la mayúscula "D", es decir, *dí-se-lo.* En cualquier caso, el problema es que se omite el acento o una mayúscula para que la palabra quede correctamente identificada y analizada.

La solución planteada es la comparación de la palabra analizada con el texto en crudo, de tal manera que, si solo se diferencia en un acento significará que es la misma palabra, y por tanto, se sobrescribirá el "*form*" de la palabra analizada por *FreeLing* con la que realmente es, es decir, con acento. Se mete en la correspondiente sección del *string* del texto en crudo dentro del *form* de la palabra analizada con setForm().

2. Join left y abreviaturas “de + el” y “a + el”.

De manera similar a la incompatibilidad anterior, *FreeLing* da como resultado de abreviaturas “*del”* o “*al*” *subtokens* con las dos partes implicadas “de” y “el” o “a” + “el” pero la norma exige un *token* para *“de”* y otro para *“l”* si la palabra es “del”, o un *token “a”* y otro *“l”* si la palabra es “al”.

Para solucionar este conflicto comprobamos primero si se cumple la regla “al” o “del” y en tal caso se sobrescribe el segundo *subtoken “el”* por *“l.* De esta manera, cuando se extrae el contenido para reflejarlo y encapsularlo en la etiqueta *token* se obtiene de manera fiel a la norma.

3. Problema con comillas simples.

En las primeras ejecuciones se descubrió que el contenido encapsulado por las etiquetas y atributos estaba escrito en comillas simples. Con este formato pueden surgir conflictos con la sintaxis XML, por eso, se reemplazaron las comillas simples por dobles en el *stringMAF* resultado.

4. Falta de etiquetas y formato de comentarios

Teniendo en cuenta las etiquetas EAGLES[30], necesarias para la interpretación del análisis morfosintáctico de *FreeLing,* se comprueba que el número de etiquetas posibles es muy elevado como para poder escribirlas manualmente. Además, la propia definición de etiquetas es incompleta, es decir, existen valores que genera *FreeLing* no recogidos en tal descripción.

Como solución a este problema realizamos un procedimiento en C++, mediante Visual Studio, por el cual se generan todas las posibles etiquetas para EAGLES. Por otro lado, se identificaron las excepciones a las reglas generales para obtener parte de las

[30] Etiquetas EAGLES para el castellano. http://nlp.lsi.upc.edu/FreeLing/doc/tagsets/tagset-es.html. Consultado en marzo de 2013.

etiquetas que faltaban, según da a esclarecer en la referencia a EAGLES. Y, por otra parte, se incluyeron una a una, aquellas excepciones particulares que no se mencionaban. Lo mismo ocurre con las etiquetas *Syn* y *Func,* y por tanto, se procedió a completarlas.

Además, una vez generadas todas las etiquetas necesarias, el formato de muestra inicial estaba desordenado, por lo que se tuvo que rehacer para mostrar cada etiqueta *comment* de manera estructurada.

5.4.2. PRUEBAS DE SISTEMA

1. Errores en el testeo del servicio web.

Se utilizó el *tester* que nos proporciona *Netbeans* que consiste de un index en lenguaje jsp que hace la llamada al método del *web Service.* Es una manera rápida de montar y probar el servicio web. No obstante, al realizar más de una ejecución se produjo errores *java.lang* sin motivo aparente. Se intentó actualizar, recompilar y desplegar de nuevo la aplicación pero el problema persistía.

La solución a este problema es el reiniciar el servidor de aplicaciones, construir el servicio y desplegarlo en el servidor reiniciado cada vez que se actualice el código fuente.

2. Error de codificación utf8 con el *tester*

Otro de los errores que tuvimos con el tester fue la codificación de los caracteres del archivo de presentación *tester.jsp*. Pese a estar configurado en UTF-8, había caracteres como la “ñ” o los acentos que no se mostraban de manera correcta.

La solución a este problema fue la creación, mediante el mismo entorno Netbeans, de un cliente con la misma funcionalidad que la clase de prueba. El cliente de prueba dio los resultados esperados, y de esta forma, comprobamos que el error era producido por la configuración por defecto del tester.

3. Creación de un cliente web de prueba que consume el servicio web.

Como el servicio web ha sido creado para ser consumido por aplicaciones cliente, la última prueba ha sido crear un cliente web orientado a los usuarios finales. Este cliente ha tenido como finalidad probar la funcionalidad del servicio web creado y mostrarlo. De esta forma, cualquier error que pueda tener el servicio web se identificará de manera sencilla, y cualquier usuario puede disponer y probar el servicio a través de la URL del cliente web.

Antes de realizar la implantación en el servidor de la Universidad, queríamos tener accesible el servicio desde la máquina virtual que replica las condiciones del servidor final.

El problema que hemos tenido con el cliente, ha sido fundamentalmente su publicación en Internet a través de la máquina virtual en la cual teníamos desplegado el servicio. Los pasos realizados para conseguirlo han sido los siguientes:

1.-Configurar el adaptador de red de la máquina virtual como 'puente'.

2.-Establecer en el host una IP fija y deshabilitar el firewall.

3.-Configurar una IP fija en la máquina virtual perteneciente a la misma red que el host y abrir el puerto que da acceso al servicio web.

4.-Configurar el router que da acceso a Internet para que redireccione las peticiones que lleguen a su IP pública a la IP privada que corresponde a la máquina virtual que tiene el servidor instalado con el servicio web disponible.

6. RESULTADOS

El resultado de nuestro proyecto es un servicio web que encapsula las funciones más importantes de *FreeLing*, es decir, el análisis sintáctico y morfosintáctico de oraciones en español, que devuelve al usuario el resultado de forma normalizada, esto es, a través de documentos XML en formato normalizado de MAF y <tiger2/> [9].

Dicho servicio web se puede integrar en cualquier otro desarrollo que quiera profundizar en este ámbito, con independencia del lenguaje de programación o plataforma tecnológica, de manera libre y gratuita. Además, si no es necesario el uso de otras funciones de *FreeLing*, los futuros desarrollos que partan de esta base se evitarán el arduo trabajo de instalar, compilar y entender dichas librerías para poder utilizarlas.

Con nuestro proyecto aportamos al mundo de las herramientas lingüísticas (caracterizadas por tratarse de desarrollos independientes, sin pautas comunes, que no permiten que las aplicaciones puedan interactuar y comunicarse) una solución que permite la interoperabilidad entre aplicaciones, facilitando la comunicación y el desarrollo de trabajos futuros, orientados a un esfuerzo común de colaboración, fomentando el uso de servicios web en el ámbito de la lingüística.

Por otra parte, la interoperabilidad no sería suficiente con el hecho de conseguir facilidad de desarrollo e integración de aplicaciones lingüísticas, sino que debe tener en cuenta la normalización de los resultados. En este sentido, nuestro proyecto apuesta por las normas del ISO/TC37, que se encarga de normalizar los principios, métodos y aplicaciones relacionadas con la terminología y las industrias de la lengua. Por este motivo, los resultados de los análisis obtenidos por el servicio web están normalizados según las normas ISO 24611:2012 (o MAF) y la ISO 24615:2010 (o SynAF).

Nuestro proyecto aporta una solución para las principales carencias de las herramientas lingüísticas, al transformarlas en un servicio web fácilmente manejable

para el procesamiento automático del lenguaje. Esta transformación facilita la personalización del software y proporciona anotaciones normalizadas. Además, el desarrollo de nuevas herramientas partiendo de esta base permite reducir el coste de un proyecto.

7. CONCLUSIONES

La realización de este proyecto ha conllevado numerosos problemas, especialmente en la primera etapa de arranque del mismo. El hecho de partir de una aplicación y unos conceptos teóricos de índole completamente diferente al entorno de estudio habitual dentro de nuestra titulación, nos ha hecho consumir la mayor parte del tiempo en entender cuál era el objetivo esperado del proyecto y cómo era el uso de la librería necesaria para llevarlo a cabo. Esto ha significado esfuerzo y tiempo enfocado al estudio e investigación del problema planteado. El análisis de la interfaz y la documentación de *FreeLing*, así como la elaboración de ejemplos prácticos sobre el mismo, ha sido crucial.

Como se ha mencionado anteriormente, *FreeLing* posee licencia GPL, lo que ha supuesto un punto de inflexión en el planteamiento del proyecto. El hecho de poder estudiar, utilizar y modificar el código nos ha permitido la creación de la envoltura lógica necesaria para la creación del servicio web. De no haber sido así, el coste y la dificultad del proyecto podrían haber sido inasumibles. Aun así, la dificultad en el análisis del código y la comprensión de su interfaz no ha sido un trabajo trivial pero sí asumible.

Esta forma de trabajar deja la puerta abierta a futuros proyectos similares que tengan como base la aplicación FreeLing. La producción de software libre, al igual que el uso de normas, permite un desarrollo colaborativo y universal que supone un gran atractivo, que esperamos atraiga la atención de futuros proyectos en este ámbito de trabajo.

Por otro lado, en el desarrollo e implementación del servicio web nos hemos encontrado con un desconocimiento práctico previo de este tipo de tecnología, lo que nos ha supuesto tiempo de estudio y trabajo. Sin embargo, una vez concluido y presentado el proyecto reconocemos el enorme valor añadido que supone encapsular las funciones de *FreeLing* en un servicio web, normalizando sus resultados. De esta forma, se da a *FreeLing* un mayor marco de uso y, sobre todo, hacemos una gran contribución con nuestro proyecto a la interoperación de los recursos lingüísticos.

8. TRABAJOS FUTUROS

A lo largo del desarrollo del proyecto se han dejado aspectos de interés que no se han podido abordar por la limitación de tiempo y contenido. Se presenta a continuación una lista de trabajos futuros o posibles mejoras del proyecto realizado:

- El servicio web creado únicamente funciona para el español; sucesivas versiones lo ampliarán a nuevas lenguas de las incluidas en *FreeLing*.
- Algunas opciones quedan pendientes de cotejar y encapsular, como son *Phonetic Encodin, o Sense Annotation.*
- Recubrimiento con el estándar LAF (Linguistic Annotation Framework) [11].
- Producir un solo archivo *<tiger2/>* con la unión de los análisis *Parser* y *Dep* que proporciona *FreeLing*. Actualmente se normaliza cada uno de ellos por separado, puesto que la estructura de sus árboles es bastante distinta. El objetivo de versiones futuras será el solapar estos dos tipos de análisis en uno solo.

9. BIBLIOGRAFÍA

[1] Criado, J. I. ; Gascó , M.; Jiménez, C. E. (Comp.) (2010) Bases para una Estrategia Iberoamericana de Interoperabilidad. Documento Marco Iberoamericano de Interoperabilidad, Ratificado en 2010 por la XX Cumbre Iberoamericana de Jefes de Estado y de Gobierno. BUENOS AIRES, ARGENTINA, JULIO de 2010.

[2] Pareja-Lora, A. (2012) *OntoTag A Linguistic and Ontological Annotation Model Suitable for the Semantic Web*. Madrid: Universidad Politécnica de Madrid (URL: http://oa.upm.es/13827/).

[3] Pareja Lora, A. (2012) *Providing Linked Linguistic and Semantic Web Annotations: The OntoTag Hybrid Annotation Model*. LAP -LAMBERT Academic Publishing, pp. 1-500.

[4] Padró, L. (Dir.) (2011) Analizadores Multilingües en FreeLing. Centro de Investigación TALP, Catalunya.

[5] Kashyap, V. , Bussler, C. y Moran, M.(2008) *The Semantic Web.* Berlín:Springer.

[6] International Organization for Standardization (2012) *ISO 24611:2012*. *Language resource management – Morpho-syntactic annotation framework (MAF)*.

[7] International Organization for Standardization (2010) *ISO 24615:2010. Language resource management – Syntactic annotation framework (SynAF)*.

[8] König, Esther; Lezius, Wolfgang. (2003) The TIGER language - A Description Language for Syntax Graphs, Formal Definition. Technical report IMS, Universität Stuttgart, Germany.

[9] Bosch, Sonja, Key-Sun Choi, Éric de La Clergerie, Alex Chengyu Fang, Gertrud Faass, Kiyong Lee, Antonio Pareja-Lora, Laurent Romary, Andreas Witt, Amir Zeldes and Florian Zipser (2012) <tiger2/> as a Standardised Serialisation for ISO 24615 – SynAF. *Proceedings of the Eleventh International Workshop on Treebanks and Linguistic Theories (TLT11)*, pp. 37 - 60. Ediçôes Colibri, Lisboa (Portugal)

[10] Institute of Electrical and Electronics Engineers (1998) IEEE Std. 830-1998 Formato de Especificación de Requisitos del Software (ERS)

[11] International Organization for Standardization (2012) ISO 24612:2012.*Language resource management – Linguistic annotation framework (LAF)*.

[12] George A. Miller (1995). WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11: 39-41. Christiane Fellbaum (1998, ed.) WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press.

[13] Fellbaum, Christiane (1998) EuroWordNet: A Multilingual Database with Lexical Semantic Networks (1998) WordNet: An Electronic Lexical Database. The MIT Press, Cambridge, MA

[14] Traductor: Google Translator. http://translate.google.es

10. APORTACIONES PERSONALES

La metodología de trabajo en el desarrollo de este proyecto ha sido fundamentalmente a través de sesiones conjuntas, en las cuales, el grupo se reunía para avanzar las tareas que en cada momento eran oportunas. En estas sesiones, nos organizábamos el reparto de actividades que en muchas ocasiones requería de dedicación personal.

APORTACIÓN DE ALICIA BALLESTEROS CALVO:

- **Búsqueda y lectura de información inicial propia del estado del arte:** al comenzar el proyecto, la primera toma de contacto fue a través de la búsqueda, lectura y estudio de distinta documentación. Por mi parte, formé parte de esta tarea plenamente que, además, posteriormente dio lugar a la sección de la memoria que corresponde al estado del arte.
- **Pruebas de instalación y configuración de *FreeLing* en Windows y Ubuntu**: una de las actividades que más tiempo nos ha llevado en el proyecto, como ya se ha explicado anteriormente, ha sido conseguir hacer funcionar *FreeLing.* Como miembro del grupo participé de todas las pruebas que se hicieron, instalando los distintos sistemas operativos, configurando las variables globales y rutas de instalación, recompilando las librerías, etc.
- **Educción de Requisitos:** a través de la documentación recogida, las reuniones que manteníamos con nuestro director y las sesiones de trabajo de nuestro grupo, fuimos identificando los requisitos del proyecto, esto es, su funcionalidad, sus restricciones y limitaciones. Igual que en las tareas anteriores, participé activamente de esta actividad.
- **Diseño y análisis:** una vez establecidos los requisitos, se debía identificar los casos de uso, las clases, preparar el entorno para la codificación, es decir, realizar el diseño de nuestro servicio web. Con la misma rutina de trabajo que anteriormente, colaboré intensamente en esta actividad.

- **Codificación:** en esta tarea me he encargado del desarrollo del algoritmo recursivo de generación de etiquetas EAGLES y los comentarios asociados a cada una de las etiquetas. Este algoritmo da como resultado un documento XML necesario para que el servicio web genere sus resultados. También he colaborado en el modelado de la normalización, en el desarrollo del servicio web y en las pruebas.
- **Implantación del servicio web en el servidor Apache dentro de una máquina virtual:** el despliegue final del servicio web ha requerido de la configuración de una máquina virtual con todas las interfaces software requeridas por el servicio web para su correcta ejecución. Debido a que la primera instalación con GlassFish se podía probar por defecto porque el entorno de programación gestionaba toda la instalación y configuración del servidor, en esta segunda instalación se requería un mayor conocimiento y trabajo para desplegar el servicio. Por este motivo, ha sido necesario buscar y estudiar manuales que nos permitieran dejar funcionando el servicio web en un servidor como Apache Tomcat 6, previamente instalado por nosotros en la máquina virtual, y debidamente configurado. En todas estas tareas he participado con mi grupo para conseguir que todo funcionara.
- **Pruebas de comunicación con el servidor web de la máquina virtual:** debido a que disponíamos de muy poco tiempo para la implantación del servicio web en el servidor de la UNED y/o UCM hemos intentado configurar la máquina virtual como un servidor web accesible desde cualquier ordenador conectado a Internet. Esta tarea nos ha llevado mucho tiempo y al igual que en el resto de tareas, también he colaborado en su realización.
- **Documentación de las distintas fases del desarrollo del proyecto:** para poder redactar toda la memoria, he ido recabando información de la documentación recogida y los pasos seguidos en las distintas etapas del desarrollo del proyecto para poder reflejarlo con la mayor claridad posible en la memoria final.
- **Realización de la memoria:** en la preparación, redacción y maquetación de la memoria he sido la principal encargada. He coordinado la recogida de información y he pedido a mis compañeros la redacción de distintas partes del documento. Me he encargado de realizar las revisiones y modificaciones oportunas que el director

del proyecto requería en cada momento. He de decir que la redacción de la memoria, aunque en un principio parece rápido y sencillo, es una labor ardua y difícil que requiere de mucho tiempo, esfuerzo y dedicación.

APORTACIÓN DE EMILIO DUOBERT COLLAZOS:

- **Traducción de los distintos documentos facilitados por el profesor para entender el desarrollo de proyecto:** como el libro The Semantic Web [5], que nos ha ayudado a entender el concepto de la Web Semántica, para qué sirve y sus aplicaciones. La traducción de las normas ISO propuestas para desarrollar el proyecto, no disponibles en español, como por ejemplo ISO/MAF [6] e ISO/SynAF [7]. Y también, la traducción de parte de la tesis de "OntoTag A Linguistic and Ontological Annotation Model Suitable for the Semantic Web" de Antonio Pareja Lora.

- **Creación de la unidad compartida para la documentación del proyecto:** he creado el servicio de alojamiento compartido para los archivos y otro servicio on-line para desarrollar la documentación de nuestro proyecto. Para el alojamiento de los archivos hemos creado una cuenta en Dropbox para guardar las diferentes versiones del proyecto y la documentación recabada. También, he gestionado Google Docs, que nos ha servido para compartir procesador de texto, hojas de cálculos, etc.
- **Obtención e instalación de los sistemas operativos para las diferentes máquinas virtuales para el desarrollo del proyecto:** Uno de las labores más arduas del proyecto ha sido conseguir instalar *FreeLing* en el sistema. He participado en la primera prueba de instalación en la plataforma Windows, utilizando diferentes versiones del sistema operativo (como XP y Windows 7). Además, participé en la segunda instalación o migración a Linux, para lo cual después de probar distintas versiones elegimos Ubuntu 12.04. que uno de los sistemas operativos más fácil de usar de la familia Linux y en la cual se hizo la compilación de FreeLing.

- **Obtención e instalación del software de desarrollo:** Para el desarrollo de nuestra aplicación sobre la plataforma Windows instalamos el entorno de desarrollo Microsoft Visual Studio ya que paralelamente al desarrollo de la aplicación íbamos haciendo pruebas de servicios web. Todo el software de Microsoft lo descargamos del sitio web de Dreamspark, el cual tiene un convenio con nuestra universidad para descargar gratuitamente el software de Microsoft que necesitemos. Al cambiar de sistema operativo (a Linux Ubuntu) también tuvimos que cambiar de entorno de desarrollo para lo cual instalamos tanto Netbeans como Eclipse, los cuales se pueden bajar desde el sitio web de Oracle y de Eclipse respectivamente, al final decidimos usar Netbeans.
- **Participación en la codificación del proyecto:** he participado durante todo el desarrollo, colaborando activamente en todo momento y he intentado estar en todas las reuniones de trabajo del grupo. En algunas ocasiones, por cuestiones laborales, he participado de las mismas gracias a las herramientas colaborativas anteriormente descritas.
- **Participación en las pruebas del proyecto:** siempre hemos intentando en todo momento la participación de todos los integrantes, sobre todo, durante el desarrollo de las pruebas del proyecto, ya que al estar todos presentes nos fue más fácil encontrar los errores cometidos, y hacer las correcciones pertinentes. El momento más crítico para todos nosotros ha sido al hacer las pruebas ya siempre encontrábamos múltiples errores por nuestra inexperiencia real en el desarrollo de proyectos, pero ha sido una experiencia muy enriquecedora.

- **Participación en la documentación del proyecto:** Todos lo realizado durante el proyecto se fue documentando, sobre todo las fuentes bibliográficas.
- **Desarrollo de la interfaz web del cliente:** he participado de esta actividad, utilizando el editor Dreamweaver para la codificación en HTML y hojas de estilo del mismo.

APORTACIÓN DE GUILLERMO CÁRCAMO ESCORZA:

- **Pruebas de instalación y configuración de *FreeLing* en Windows y Ubuntu**: participé activamente de todas las pruebas que se hicieron en esta actividad. Tras dos meses con este problema, finalmente conseguí que arrancara.

- **Instalación de un programa de conexión remota:** Muchas veces, para comprobar lo que estábamos desarrollando cada uno de nosotros, y al encontrarnos en diferentes ubicaciones físicas, necesitábamos flexibilidad en el desarrollo. Por esta razón, instalamos *TeamViewer8.* Este programa, aparte de controlar los escritorios de forma remota, también nos permitió realizar videoconferencias entre los integrantes del grupo, permitiendo la simultaneidad en la elaboración del código y una comunicación flexible entre nosotros. Otras de las ventajas más importantes de este programa es que se puede instalar en diferentes sistemas operativos, y al estar utilizando tanto Linux para la máquina virtual de VMWare, como Windows para los retoques HTML del cliente, esto nos supuso una ventaja.
- **Diseño y análisis:** dentro de esta tarea, elaboré un modelo, con la ayuda de mis compañeros, que ha servido de unión entre *FreeLing* y la normalización pedida. Este proceso de estudio de las normas, así como de la interfaz y el código de los módulos de *FreeLing* ha servido para extraer este modelo.

- **Implementación del Servidor Web:** después de probar en varios servidores de aplicaciones Web como Apache y GlassFish, tuvimos que instalar un servidor de prueba para nuestra aplicación. Debido a que el servidor final en el que se alojará el servicio utiliza Apache Tomcat 6, hicimos una réplica en nuestra máquina virtual, es decir, emulamos las condiciones del servidor final que nos proporciona nuestro tutor en nuestra máquina virtual VMWare haciendo conexión de puente con la máquina anfitrión, que posee Windows 7.

- **Desarrollo de la interfaz web del cliente:** Para el desarrollo de la interfaz web del cliente que hace las peticiones al servicio web se ha utilizado codificación en HTML, hojas de estilo CSS lo cual se puede hacer en cualquier editor de texto, con la experiencia que tuvimos en diversos asignaturas del grado nos decidimos usar el programa *notepad++* que tiene muchas ventajas entre las que cabe destacar que es gratuito y soporta múltiples lenguajes de programación (sobre todo es muy utilizado en el desarrollo de páginas webs), también tiene muchas opciones avanzadas para desarrolladores, como un potente entorno de desarrollo con la sencillez de un block de notas. Para la parte dinámica de la página se utilizó codificación *.jsp* que es la parte del código por la cual solicitará y pasará datos al servicio web implementado.

- **Obtención del Hosting:** Para alojar nuestra aplicación a través de nuestro tutor hemos conseguido el alojamiento en el servidor de la UNED, para poder implementar nuestra aplicación antes tuvimos que hacer múltiples pruebas en nuestra réplica del Servidor Web (Apache Tomcat6) ya que el de la universidad es un servidor que está en producción y el que se tiene que manipular con mucho cuidado.
- **Codificación de la aplicación:** en esta actividad he participado de manera activa y he liderado el progreso de la misma, debido a que soy el integrante del grupo que mejor conozco la codificación en Java. Yo he delegado las tareas a desarrollar por cada uno de los integrantes del grupo. También tuve que hacer las correcciones y el montaje de los diferentes módulos desarrollados por cada uno de nosotros.

- **Ayuda en la elaboración de le memoria final y correcciones**: al ser el coordinador de la parte de codificación del proyecto, he sido el encargado de redactar la parte más específica de desarrollo y codificación de la memoria.