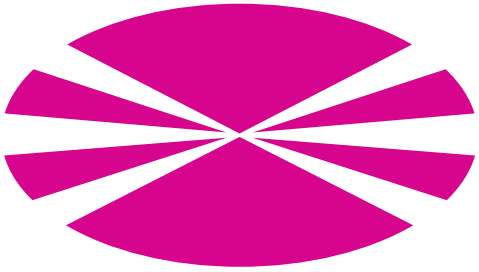


Facultad de Informática de la Universidade da Coruña
Departamento de Computación

PROYECTO FIN DE CARRERA
INGENIERÍA INFORMÁTICA

Desarrollo de una Interfaz de Lenguaje Natural para una Base de Conocimiento Online

Alumno: Yerai Doval Mosquera

Directores: Jesús Vilares Ferro
Carlos Gómez Rodríguez
Iván Gómez Caderno

Fecha: 21 de junio de 2013

Resumen

El acceso a los datos estructurados contenidos en una base de conocimiento requiere del uso de lenguajes formales de consulta, lo cual puede constituir una importante limitación para los usuarios no expertos.

Este problema puede ser resuelto mediante una interfaz de lenguaje natural en la que el usuario pueda expresar las consultas de una manera mucho más sencilla, tal y como si hiciese una pregunta a otra persona.

El objetivo del proyecto consiste, pues, en interpretar una pregunta escrita en lenguaje natural y traducirla al lenguaje de consulta del sistema correspondiente.

Para la presentación del proyecto se creará una página web, con una caja de texto similar a la del buscador de Google, en la que el usuario podrá introducir su consulta y visualizar directamente la respuesta, obtenida de la base de conocimiento.

Palabras clave

Interfaz de Usuario, *Interfaz de Lenguaje Natural*, *Procesamiento del Lenguaje Natural*, *Base de Conocimiento Online*

Software utilizado

Las herramientas utilizadas para el desarrollo del proyecto, agrupadas por tipo, son las siguientes:

- *Lenguaje de programación*
  - Java
- *IDE*
  - NetBeans
- *Frameworks de programación*
  - Hibernate
  - Spring
  - Struts 2
  - Logback
  - JUnit
- *Gestión de proyectos*
  - Subversion
  - Apache Maven
- *Gestor de base de datos*
  - MySQL
  - MySQLWorkbench (herramienta gráfica de gestión de BBDD)
- *Servidor web*
  - Jetty
- *Herramientas de NLP*
  - Apache OpenNLP
  - TreeTagger
  - Google Spell Check
- *Clasificadores*

    - Weka
- *Entorno de ejecución*
    - Vagrant
        - VirtualBox
        - Ubuntu Server 12.04.02 LTS 64 bits

A todos mis amigos, y a mi familia en especial.

Agradecimientos

Mi más sincero agradecimiento a mis directores, Jesús, Carlos e Iván, por su inestimable apoyo tanto técnico como moral, por estar disponibles siempre que requerí de su ayuda y, en resumidas cuentas, por hacerme sentir tan cómodo durante el desarrollo del proyecto, a pesar del gran desafío que supuso.

Por otra parte, agradecer también la ayuda recibida de los miembros del grupo COLE y del grupo LYS, los cuales contribuyeron en este proyecto con un corpus de preguntas.

Índice general

Índice de figuras

Parte I

¿Qué son las Interfaces de Lenguaje Natural?

Capítulo 1

Introducción

Una Interfaz de Lenguaje Natural (ILN) es cualquier capa de software o hardware que permite a un usuario interactuar con un determinado sistema utilizando un idioma humano, tal como el inglés, el gallego o el español. De esta forma la interacción entre el usuario y el sistema se lleva a cabo de una manera mucho más sencilla y natural para el primero.

Tradicionalmente, para interactuar con sistemas tales como bases de datos relacionales, es necesario el uso de un lenguaje formal (artificial), como es el caso de SQL, lo cual limita seriamente la accesibilidad a este tipo de sistemas por parte del gran público. Si bien cualquier persona podría aprender a usar este lenguaje de consulta para interactuar con la base de datos, esto requeriría de un período de formación adecuada que actúa a su vez también a modo de barrera. Como respuesta a esta situación surgen las interfaces de consulta. Sin embargo, dichas interfaces son por lo general limitadas, poco flexibles o requieren a su vez de un período de aprendizaje para su uso. Lo que se pretende con las interfaces de lenguaje natural es precisamente superar tales limitaciones, de forma que el usuario interactúe de manera efectiva con el sistema de un modo sencillo y natural para él empleando para ello su propio idioma.

1.1. Contextualización: Classora Knowledge Base y la Web Semántica

Si bien hasta ahora hemos mencionado las bases de datos, en el contexto de este trabajo vamos más allá, al tratar con *bases de conocimiento*. Para ver las diferencias entre estas dos entidades, primero necesitamos entender las diferencias entre *datos* y *conocimiento*.

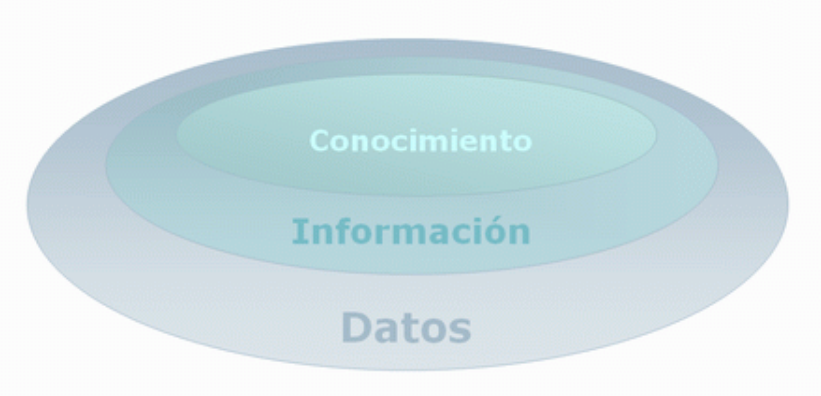


Figura 1.1: Datos, información y conocimiento.

Un ejemplo muy claro para ver la diferencia entre datos, información y conocimiento, es el de la *regla del teléfono* [Classora, 2012]. Un número de teléfono por sí solo representa un *dato*, mientras que un listín telefónico convenientemente organizado y estructurado representa *información*, y todo el procedimiento necesario para localizar el número que se necesita, teclearlo en el móvil, y contactar con el interlocutor deseado, constituiría el *conocimiento*. En la Figura 1.1 se puede ver, de forma gráfica, la relación existente entre estos tres conceptos.

Las *bases de conocimiento* nacieron como una evolución de las bases de datos, fruto de una mayor documentación de los elementos que contienen. Esto resulta posible gracias a la asociación de *metadatos* que describan estos elementos en términos fácilmente entendibles tanto por humanos como por máquinas (a través de algún sistema informático) [Classora, 2012]. Hoy en día existen iniciativas abiertas y disponibles a través de la Web tales como Wolfram Alpha, DBpedia, Freebase o Evi.[1]

En el presente proyecto trabajaremos en concreto con la base de conocimiento desarrollada por Classora Technologies: Classora Knowledge Base (CKB) [CKB, 2013], la cual también está disponible online a través de la Web. CKB se diferencia del resto de alternativas en la aplicación de técnicas de Business Intelligence, lo que permite cruzar información de cada unidad de conocimiento con todos los informes en los que figura, monitorizar su evolución temporal, y representar todos estos resultados en diferentes formatos [Classora, 2012].

Las iniciativas expuestas anteriormente forman parte de lo que se denomina como *Web Semántica*. La Web Semántica es una evolución de la Web tradicional, en la que los usuarios

[1] http://www.wolframalpha.com, http://es.dbpedia.org, http://www.freebase.com, http://www.evi.com, respectivamente.

son capaces de encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a una información mejor definida. Debido a esto último, el software es capaz de procesar su contenido, razonar con éste, combinarlo y realizar deducciones lógicas para resolver problemas automáticamente. Para obtener esa definición adecuada de los datos, así como interactuar con ellos, la Web Semántica utiliza lenguajes formales como RDF, SPARQL u OWL [W3C, 2013].

1.2. Motivación

Los lenguajes de representación del conocimiento, es decir, aquellos que permiten generar la información estructurada que se puede encontrar en las bases de conocimiento, se han ido popularizando de forma notable durante los últimos años. Esto ha provocado un aumento de la información estructurada accesible a través de Internet, tratándose por lo tanto de recursos fácilmente accesibles por el gran público, en el que se incluyen tanto usuarios expertos como casuales. Así, este último grupo de usuarios requiere que el acceso a estos recursos se pueda realizar de la manera más cómoda e inmediata posible. Se hace necesario, pues, el desarrollo de interfaces que faciliten la interacción con el sistema por parte de los usuarios. Sin embargo, tales interfaces suelen ser por lo general limitadas, poco flexibles o requerir a su vez de una cierta formación por parte del usuario inexperto. Es aquí donde entran en juego las interfaces de lenguaje natural, ya que permiten abstraer la complejidad de los lenguajes formales de consulta realizando una traducción entre ambos tipos de lenguajes (ver Figura 1.2). De esta forma, si quisiéramos conocer la fecha de nacimiento de Fernando Alonso, por ejemplo, tan sólo tendríamos que escribir:

```
Dime cuál es la fecha de nacimiento de Fernando Alonso.
```

y sería la propia interfaz la encargada de traducir esta formulación inicial de la consulta en lenguaje natural a, pongamos por caso, una consulta SQL para interrogar una base de datos que almacenase tal información:

```
SELECT fecha_nacimiento FROM personas WHERE nombre='Fernando Alonso';
```

De acuerdo con un estudio realizado por [Kaufmann and Bernstein, 2007], los usuarios suelen preferir aquellos sistemas que soportan interfaces de lenguaje natural. Se concluyó además que las consultas en forma de enunciado eran preferidas a aquellas basadas en palabras clave. Esta última afirmación podría matizarse, tal y como se concluye de los resultados de la evaluación del sistema CHESt [Linckels and Meinel, 2007]. Un 22 % de los usuarios encuestados afirmó preferir las consultas de enunciado a las de palabras clave,

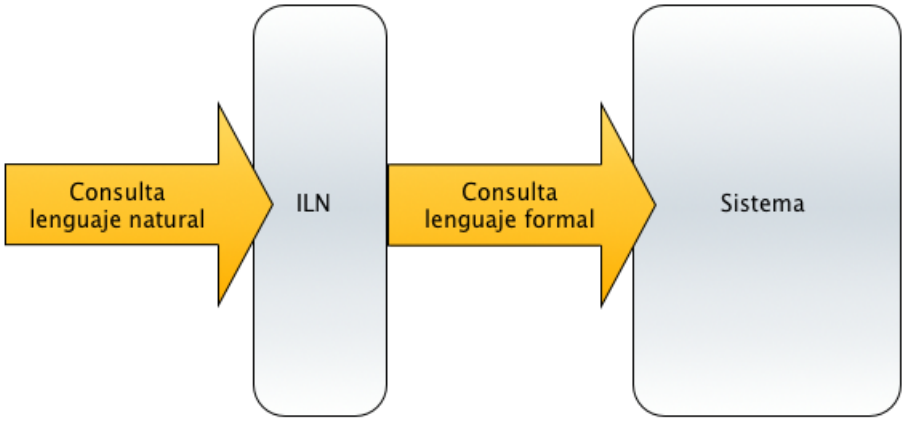


Figura 1.2: Esquema general.

mientras que un 69 % observó que únicamente lo preferiría si los resultados fueran mejores con las consultas en forma de enunciado.

Por otra parte, tal y como se concluye del trabajo de [Iskold, 2008], la web semántica nos ayudaría a encontrar las respuestas a consultas complejas como si se tratara de una base de datos. Sin embargo, para expresar estas consultas, el uso de palabras clave resulta insuficiente, requiriéndose el empleo de expresiones más complejas propias del lenguaje natural.

1.3. Objetivos

Para el desarrollo de este trabajo, es necesario familiarizarse primero con la estructura interna de la propia base de conocimiento. Al tratarse en este caso de una base de conocimiento empresarial de ámbito cerrado, nuestro acceso a dicha estructura ha sido restringido, por razones obvias. Sin embargo, el subconjunto expuesto para la realización de este proyecto es suficiente para los propósitos perseguidos, y está formado por los siguientes conceptos:

- *Unidad de conocimiento.* Tipo de entidad sobre la que se guarda conocimiento (p.ej. persona, empresa, país, etc).

- *Atributo.* Conocimiento que se guarda sobre los tipos de entidades (p.ej. para persona: edad, fecha de nacimiento, etc).

Para la consulta de los datos contenidos en CKB es necesario disponer de herramientas que guíen al usuario y faciliten el acceso a toda la información disponible. En este sentido Classora Technologies ha desarrollado un lenguaje semiformal muy básico, denominado Classora Query Language (CQL), que actúa como base para la comprensión de los requisitos del usuario. En esta primera aproximación, el lenguaje sirve para hacer dos tipos de consulta (los ejemplos se muestran junto con sus equivalentes en lenguaje natural):

- *Tipo 1.* Obtener atributos de una unidad de conocimiento:

  ```
  facturación, microsoft
  ```

  Dime cuál es la facturación de la empresa Microsoft.

  ```
  fecha de nacimiento, barack obama
  ```

  Dime cuál es la fecha de nacimiento de Barack Obama.

- *Tipo 2.* Obtener unidades de conocimiento que satisfacen una condición:

  ```
  edificio*, ubicación = estados unidos, altura < 300 metros
  ```

  Encuentra todos los edificios construídos en Estados Unidos que midan más de 300 metros de altura.

El objetivo último de este proyecto es el diseño, desarrollo y despliegue en forma de aplicación web, de una interfaz de lenguaje natural para Classora Knowledge Base, en la que se realice la traducción entre la consulta del usuario, formulada en lenguaje natural, y su consulta equivalente en CQL, que será enviada a la base de conocimiento. Una vez procesada la consulta, el usuario podrá visualizar la respuesta a su consulta desde esta misma interfaz.

Si bien es parte de los requisitos del proyecto que el sistema se pueda adaptar para el uso de diferentes idiomas de entrada (además del español), se ofrecerá además una solución que también soporte diferentes lenguajes formales de salida (además de CQL). Sin embargo, no se considerará la portabilidad del sistema a otras bases de conocimiento distintas, al menos no de forma directa.

Como consecuencia de todo esto, y como uno de los objetivos del proyecto, se busca proveer al usuario de un acceso más natural a la información contenida dentro de CKB. En general, lo que se pretende con este tipo de sistemas es ocultar al usuario la complejidad de la estructura de la base de conocimiento subyacente y el acceso a la misma.

1.4. Retos

Los retos que se nos presentan a la hora de construir interfaces de lenguaje natural están directamente relacionados con las características inherentes al lenguaje humano. De este modo, para implementar un sistema robusto habremos de lidiar con la *ambigüedad* y la *expresividad* del lenguaje natural [Damljanović, 2011].

Se dice que un enunciado escrito en lenguaje natural es *ambiguo* cuando para ese mismo enunciado existe más de una interpretación de su significado. Esto representa un problema puesto que supondría que para una misma entrada al sistema tendríamos varias salidas. Una posible solución para filtrar estas interpretaciones sería mantener cierta información sobre la relevancia de los términos aparecidos en cada una de ellas, prefiriendo siempre la interpretación que contenga los términos de mayor relevancia. El inconveniente de esta aproximación es que no se tiene en cuenta la opinión del usuario. Así, una segunda aproximación, que bien se podría usar en conjunción con la primera, es provocar un diálogo con el usuario en el que éste pueda validar o elegir la interpretación. Por otra parte, existen los llamados *lenguajes naturales controlados*,[2] que también nos permitirían lidiar con el problema de la ambigüedad [Fuchs et al., 2006]. El gran inconveniente de ésta última solución es que tales lenguajes, si bien se parecen en gran medida a los lenguajes naturales, requieren que el usuario pase por una etapa de aprendizaje, y esto es precisamente lo que queremos evitar.

La *expresividad* de un lenguaje natural, por su parte, se refleja en que varios enunciados distintos pueden tener el mismo significado (misma interpretación). Esta cualidad afecta directamente a la robustez del sistema, puesto que se ha de asegurar que para tales entradas distintas se obtengan las mismas salidas. Para resolver este problema se suelen hacer operaciones sobre la entrada al sistema de cara a *normalizarla* de algún modo. Así, por ejemplo, para soportar todas las *inflexiones morfológicas* que puedan tener las palabras de un enunciado, las interfaces de lenguaje natural suelen operar sobre los *lemas*[3] de estas palabras, y no directamente sobre las formas. Los *sinónimos*, a su vez, podrían tratarse mediante el uso de una herramienta tal como WordNet [Fellbaum, 1998] o un diccionario de sinónimos [Fernández et al., 2003].

Como respuesta a estos fenómenos, también denominados de forma conjunta *variación lingüística* [Vilares, 2005], es necesaria la aplicación de técnicas de procesamiento del lenguaje natural [Jurafsky and Martin, 2009, Dale et al., 2000] siendo ésta la rama de las

[2]Lenguajes basados en su propia sintaxis bien definida y sin ambigüedades, y con los que se puede construir un subconjunto restringido del total de las expresiones que se podrían obtener al usar un lenguaje humano.

[3]Definimos *lema* como la forma canónica de una palabra o, de forma más sencilla, su correspondiente entrada en un diccionario. Por ejemplo “guapo” para “guapas” o “comer” para “comamos”.

ciencias computacionales encargada del diseño e implementación de los elementos software y hardware necesarios para el tratamiento del lenguaje humano.

1.5. Organización del proyecto

El presente trabajo se encuentra dividido en cuatro partes bien diferenciadas, divididas cada una en capítulos.

- *¿Qué son las Interfaces de Lenguaje Natural?*. Esta será la pregunta a la que se intentará responder en esta primera parte, sirviendo de toma de contacto con el contexto en el que se desarrolla el resto del trabajo. Esta parte incluye el siguiente capítulo:
  - *Introducción*. Presentación del ámbito de trabajo. Se expondrán aspectos como la motivación del proyecto, el contexto en el que se desarrolla, los objetivos del mismo y los retos que se han de superar durante su construcción.
- *Fundamentos Tecnológicos y Estado del Arte*. Se realizará en esta parte una exposición de las bases teóricas del proyecto abordado, de otros proyectos o herramientas existentes similares al aquí desarrollado y, por último, un estudio de la viabilidad del sistema. Esta parte está constituida por los siguientes capítulos:
  - *Fundamentos teóricos*. Descripción de las bases teóricas sobre las que se fundamenta nuestro sistema.
  - *Análisis de antecedentes y alternativas*. Estudio comparativo de sistemas de funcionalidad similar.
  - *Estudio de viabilidad*. Descripción de las herramientas empleadas para el desarrollo de este sistema.
- *Construyendo Interfaces de Lenguaje Natural*. En esta parte se explicará todo el proceso seguido a la hora de desarrollar nuestro sistema. Se divide en los siguientes capítulos:
  - *Metodología*. Descripción de los aspectos metodológicos relativos a análisis, diseño e implementación del sistema desarrollado.
  - *Ejemplos de funcionamiento*. Estudio de dos ejemplos representativos de uso del sistema con el fin de observar su funcionamiento interno.
  - *Pruebas realizadas*. Análisis del proceso y resultados de la verificación y validación del sistema.

  - *Gestión del proyecto.* Exposición del plan seguido para el desarrollo del sistema y gestión de riesgos.

- *Resultados y Conclusiones.* Finalmente, presentaremos un análisis del trabajo realizado y de los resultados obtenidos. Esta última parte contiene los siguientes capítulos:

  - *Análisis de los resultados* obtenidos y exposición de las ampliaciones o mejoras que se podrían desarrollar en un futuro.
  - *Conclusiones* a las que se ha llegado tras el desarrollo del sistema.

Parte II

Fundamentos Tecnológicos y Estado del Arte

Capítulo 2

Fundamentos teóricos

Este trabajo se apoya en los fundamentos y las ideas propuestas en otros sistemas de propósito similar, integrándolos y extendiéndolos de forma conveniente para conseguir la funcionalidad deseada. Así, como base fundamental se han considerado los sistemas de Búsqueda de Respuestas y las Interfaces de Lenguaje Natural, además de las técnicas de mapeo semántico.

2.1. Sistemas de Búsqueda de Respuestas

De acuerdo con [Prager, 2006], la *Búsqueda de Respuestas* (BR) es el campo de estudio que concierne al desarrollo de sistemas automáticos que generan respuestas a preguntas formuladas en lenguaje natural a partir de bases documentales. De este extenso campo se pueden extraer un buen número de conceptos y técnicas aplicables a nuestro sistema, ya que normalmente las interfaces de lenguaje natural aparecen formando parte de un sistema de BR. El estudio de los sistemas de BR se remonta hasta los años 60 [Simmons, 1965]. Medio siglo después, la construcción de sistemas que sean capaces de responder automáticamente preguntas tal y como lo haría un humano sigue siendo uno de los mayores desafíos del mundo de la informática.

2.1.1. Clasificaciones de los sistemas de BR

Existen varias formas de clasificar los sistemas de BR. Atendiendo a la amplitud del dominio de la información usada por el sistema para extraer respuestas, se puede distinguir entre dos tipos de sistemas:

- *Sistemas de dominio abierto.* El dominio de la información disponible es muy amplio, cubriendo varios campos de conocimiento y permitiendo prácticamente cualquier tipo de consulta. Sin embargo, esto no garantiza que la respuesta se encuentre entre la información disponible, o que ésta, en cualquier caso, sea la correcta. MURAX [Kupiec, 1993] fue uno de los primeros sistemas de este tipo.
- *Sistemas de dominio cerrado.* En este caso, el dominio de la información disponible es restringido, permitiendo únicamente las preguntas versadas sobre ciertos conceptos de un determinado campo de conocimiento. La ventaja de este tipo de sistemas radica precisamente en que permite al sistema especializarse en un campo de conocimiento y ofrecer así mejores respuestas a las preguntas soportadas.

Desde esta perspectiva, el sistema desarrollado podría englobarse dentro de un sistema de BR de dominio cerrado, puesto que la información disponible y su estructura están determinados por la base de conocimiento sobre la que se opera.

Por otra parte, los sistemas de BR se pueden clasificar en función de las técnicas de PLN que utilizan [Vicedo, 2003]. Dado que nuestro sistema también hace uso de dichas técnicas, tiene sentido tratar de clasificarlo bajo estos mismos términos:

- *Sistemas de clase 0.* Este tipo de sistemas de BR no utilizan ninguna técnica de PLN.
- *Sistemas de clase 1.* Estos sistemas realizan análisis léxico-sintáctico durante el procesamiento de la consulta.
- *Sistemas de clase 2.* Añaden el análisis semántico al procesamiento de la consulta.
- *Sistemas de clase 3.* Añaden a los anteriores un análisis contextual.

Según esta clasificación, el sistema desarrollado podría englobarse dentro de un sistema de BR de *clase 2*, ya que no sólo se aplican técnicas de PLN para el análisis léxico de las consultas, sino que se añade además un análisis semántico basado en representaciones internas de los términos de interés que puedan aparecer en una consulta. Estas representaciones estarán formadas por el término en sí, datos e información descriptiva del mismo (metadatos), y las relaciones con otros términos.

2.1.2. Terminología de los sistemas de BR

A continuación se expone un subconjunto de la terminología de los sistemas de BR relevante para el contexto de nuestro sistema [Prager, 2006]:

- *Tipo de pregunta*. Categorización de la pregunta, la cual permitirá seleccionar la estrategia de procesamiento más adecuada a la misma, así como el formato de la respuesta. En el contexto del sistema desarrollado en este trabajo, distinguiremos entre preguntas sobre *hechos*, también denominadas *preguntas factuales* ("¿Cuántos años tiene Fernando Alonso?") y *listas* ("¿Qué presidentes tuvo España?"). Existen además muchos otros tipos de pregunta que no ha sido necesario considerar en nuestro sistema pero que sí son abordadas por algunos sistemas de BR: definición, relación, opinión, causa-efecto, etc.

- *Tipo de respuesta*. Se trata del tipo de objeto buscado en la pregunta. Por ejemplo, las preguntas que empiezan por *quién* harán referencia a una persona, las que empiecen por *cuándo* a una fecha, etc.

- *Foco de la pregunta*. Es el *qué* buscamos conocer al formular la pregunta. Se encuentra íntimamente relacionado con el *tipo de respuesta*, en tanto que suele tratarse de una *instancia* de la clase definida por éste. En la pregunta "¿Cuántos años tiene Fernando Alonso?", el foco es "años", que es un valor numérico (tipo de respuesta).

- *Tema de la pregunta*. El *tema* hace referencia a la *entidad* (persona, lugar, ...) sobre la que se realiza la pregunta. En el ejemplo anterior, "¿Cuántos años tiene Fernando Alonso?", la entidad Fernando Alonso es el tema de la pregunta.

- *Lista de autoridad*. Se trata de una colección de instancias de un determinado tipo de respuesta (clase) de interés. Un ejemplo podría ser una lista con nombres de pilotos de Fórmula 1.

2.2. Interfaces de Lenguaje Natural

Desde el punto de vista de los usuarios finales, el lenguaje natural permite una interacción más eficiente con un sistema informático [Ogden and Bernick, 1997]. Las ILN se construyen por diversos motivos, aunque la mayoría de ellos tienen relación con el problema del acceso al conocimiento almacenado en uno de estos sistemas. Así, se puede realizar una clasificación de las *Interfaces de Lenguaje Natural* (ILN) de acuerdo con la estructura del conocimiento subyacente a la que se desea acceder [Damljanović, 2011]:

- ILN a datos estructurados.

- ILN a datos semi/desestructurados.

- ILN interactivos.

Conviene destacar aquí que no todas las ILN son desarrolladas para el acceso a conocimiento. Otro campo en el que también se estudió el uso de lenguajes naturales es en el de la programación [Biermann et al., 1983], por ejemplo.

2.2.1. ILN a datos estructurados

Para la interacción con ciertos tipos de sistemas, tales como sistemas de bases de datos, un usuario requiere del uso de lenguajes formales de consulta. La intención al implementar ILNs es permitir al usuario comunicarse con estos sistemas a través del uso del propio lenguaje humano, evitando así el proceso de aprendizaje del lenguaje formal. Estas interfaces suelen aparecer formando parte de sistemas de dominio cerrado, y se dividen a su vez en dos subgrupos: ILN a bases de datos relacionales e ILN a ontologías.

2.2.1.1. ILN a bases de datos relacionales

Estas interfaces traducen las consultas en lenguaje natural al correspondiente lenguaje de consulta, SQL por lo general, permitiendo la extracción de respuestas de una base de datos relacional. La mayor parte de las ILNs a datos estructurados desarrolladas hasta el momento pertenecen a este grupo. De entre las primeras en aparecer, alrededor de mediados de los años 60 y principios de los 70, la más popular es LUNAR [Woods et al., 1972], desarrollada para trabajar sobre una base de datos acerca del análisis químico de rocas lunares.

Atendiendo a la forma de realizar la traducción y su arquitectura interna, estos sistemas se dividen en varios tipos [Damljanović, 2011]:

- *Sistemas basados en correspondencia de patrones (pattern-matching).* Son fáciles de implementar y efectivos para consultas simples, pero insuficientes para consultas complejas, donde es muy posible que se devuelvan respuestas incompletas debido a que los patrones establecidos pueden no cubrir todas las posibilidades de formulación de las consultas soportadas por el sistema. Por ejemplo, para la pregunta "¿Quién es el presidente de EEUU?", si el sistema no posee el patrón para "presidente" seguido de un país y sólo posee "presidente", la respuesta a esta pregunta serán todos los presidentes que aparezcan en la base de datos.

- *Sistemas basados en sintaxis.* En este caso, el árbol sintáctico extraído de la consulta es directamente traducido al lenguaje formal. Requieren, por lo tanto, de la aplicación de técnicas de análisis sintáctico [Jurafsky and Martin, 2009, Ch. 13, 14].

- *Sistemas basados en gramática y semántica.* Estos sistemas también construyen un árbol de análisis a partir de la consulta, sólo que en este caso los nodos no terminales se corresponden con conceptos semánticos (representados internamente en el propio sistema) y no sintácticos como en el caso anterior. La desventaja es que se dificulta la portabilidad del sistema.

- *Sistemas con lenguaje de representación intermedio.* Estos sistemas traducen primero la consulta en lenguaje natural a un lenguaje de representación intermedio, independiente de la estructura de la base de datos. Esta aproximación solventa el problema de portabilidad del caso anterior, puesto que el lenguaje de representación intermedio puede ser luego traducido al lenguaje formal concreto de la base de datos con la que se trabaje, sin la necesidad de rediseñar todo el sistema.

2.2.1.2. ILN a ontologías

En el campo de la informática, el término *ontología* hace referencia al esquema lógico de roles y conceptos y las relaciones existentes entre ellos [Antoniou and Van Harmelen, 2004]. Por otra parte, *base de conocimiento* hace referencia al conjunto de instancias o individuos generados a partir de las definiciones de la ontología. Estos dos conceptos se relacionan entre sí de la misma manera que lo hace un esquema de base de datos relacional con los datos almacenados según dicho esquema. Para la construcción y explotación de estas bases de conocimiento existen lenguajes formales ampliamente utilizados, como OWL o RDF para la especificación de ontologías, y SPARQL como lenguaje formal de consulta más utilizado.

La ventaja que presentan las ontologías sobre los esquemas de base de datos, respecto a la construcción de una ILN que trabaje sobre una de estas estructuras, es precisamente el elemento diferencial entre la una y la otra: el enriquecimiento semántico de los datos. Cuanto mejor definidos (semánticamente enriquecidos) se encuentren los conceptos en la ontología, más fácil nos resultará encontrar correspondencias exactas en la base de conocimiento a los términos que aparezcan en las construcciones del lenguaje natural. Como contrapartida, cabe mencionar que la excesiva especialización de una ILN para trabajar con una determinada ontología (sacando así mayor partido a sus cualidades específicas) puede comprometer la portabilidad de dicha interfaz a otras ontologías.

Algunos ejemplos de este tipo de sistemas son Querix [Kaufmann et al., 2006], AquaLog [Lopez and Motta, 2004], PowerAqua [Lopez et al., 2012], o el propio sistema desarrollado en este proyecto.

2.2.2. ILN a datos semi/desestructurados

Se distinguen del grupo anterior en que no necesitan traducir la consulta en lenguaje natural a ningún lenguaje formal, puesto que procesan directamente la colección de documentos (archivos de noticias, preguntas frecuentes, etc) sin la necesidad de interactuar con un sistema gestor de dicha colección. Los sistemas más prominentes dentro de este grupo son los que conforman los sistemas de BR de dominio abierto, los cuales procesan enormes cantidades de documentos para encontrar las respuestas requeridas. Conviene distinguir este tipo de sistemas de los sistemas de Recuperación de Información tales como el buscador de Google, los cuales buscan una lista de *documentos relevantes* a la consulta del usuario y los devuelven como respuesta. En contraste, los sistemas de BR buscan la *respuesta* concreta a la consulta del usuario, que probablemente se encontraría en uno de estos documentos relevantes. Algunos ejemplos son MURAX [Kupiec, 1993], MULDER [Kwok et al., 2001] o AnswerBus [Zheng, 2002].

Otro tipo de sistemas que también pertenece a este grupo es el de los sistemas de Comprensión de Lectura. Sistemas como Deep Read [Hirschman et al., 1999] son capaces de encontrar las respuestas a un conjunto de preguntas relacionadas con una historia escrita en lenguaje natural. Este tipo de sistemas suele usarse para probar el nivel de lectura en niños.

2.2.3. ILN interactivas

En muchas ocasiones, los sistemas de BR de dominio abierto no son capaces de responder preguntas complejas formuladas de forma aislada [Webb and Webber, 2009]. En estas preguntas suelen aparecer hipótesis y sus consecuencias, analogías y comparaciones, etc., construcciones que no sería posible analizar sin la existencia de un *contexto* de conversación. Así es como surgen las *ILN interactivas*, que involucran al usuario en un diálogo con el sistema cuyo contexto se irá formando conforme las interacciones pregunta–respuesta se sucedan. Las preguntas lanzadas por el usuario no son consideradas independientes entre sí, de modo que la memorización e interpretación de las entradas previas, junto con las respuestas ofrecidas, permiten aumentar la capacidad del sistema para responder preguntas posteriores. La gran dificultad en el desarrollo de este tipo de sistemas es cómo aislar los contextos de diferentes conversaciones.

Uno de los primeros sistemas de este tipo data de mediados de los años 60, principios de los 70: SHRDLU [Winograd, 1972]. A pesar de que la interacción con este sistema se llevaba a cabo a través de una interfaz de texto, sistemas más recientes implementan, con ciertas limitaciones, interfaces de conversación por voz [Allen et al., 1996].

Si bien en la mayoría de los sistemas de este tipo no existe una meta a alcanzar con el diálogo usuario–sistema, desde el punto de vista del propio sistema, existen también los llamados sistemas de *tutoría*. Estos sistemas tienen como objetivo ayudar a los estudiantes en los procesos de aprendizaje y corregir sus errores, siendo el más conocido de ellos ITSPOKE [Litman and Silliman, 2004].

2.3. Mapeo semántico

Si bien ya hemos hablado acerca del uso de información semántica en el caso de los sistemas de BR (sistemas de clase 2 y 3) y de las ILN a datos estructurados (sistemas semántico–gramaticales), conviene hacer un análisis más profundo del concepto de *mapeo semántico*, uno de los pilares sobre los que se sustenta el sistema desarrollado.

Las ILN suelen usar dos tipos de lenguajes de entrada: lenguaje natural controlado (o restringido) y lenguaje natural no restringido. Si bien el segundo tipo engloba el lenguaje humano, al primero pertenecen aquellos lenguajes basados en su propia sintaxis bien definida y sin ambigüedades, y con los que se puede construir un subconjunto restringido del total de las expresiones que se podrían obtener al usar un lenguaje humano. Estos lenguajes tienen la particularidad de que se pueden traducir directamente y sin ambigüedades a diversos lenguajes formales de lógica de primer orden. El ejemplo más conocido es Attempto Controlled English (ACE) [Fuchs et al., 2006].

En el caso del lenguaje humano no restringido, donde las expresiones ambiguas, vagas y potencialmente inconsistentes se dan con frecuencia, es necesario establecer algún mecanismo que permita una traducción automatizada al lenguaje formal deseado. A este respecto existen principalmente dos aproximaciones para la obtención de dicha traducción: *basada en datos* y *basada en léxico* [Gao et al., 2011].

2.3.1. Aproximación basada en datos

En esta aproximación, el proceso de traducción se basa en los datos obtenidos a partir del análisis de la consulta y en cierta información almacenada en el sistema en forma de patrones. Podemos a su vez hablar de:

- *Correspondencia de patrones (pattern-matching)*. LUNAR [Woods et al., 1972] es un claro ejemplo de este tipo de sistemas, de los que ya hemos hablado en el Apartado 2.2.1.1,

- *Mapeo basado en aprendizaje.* Se basa en la obtención de patrones mediante un proceso previo de aprendizaje automático a partir de un conjunto de entrenamiento de pares de la forma `(consulta en lenguaje natural, consulta en lenguaje formal)`. Su principal inconveniente es que se necesita un conjunto de entrenamiento muy extenso para que este tipo de sistemas resulten efectivos. Un ejemplo ilustrativo de este caso es QACID [Ferrández et al., 2009].

2.3.2. Aproximación basada en léxico

En esta ocasión el proceso de traducción extrae datos de la consulta formulada por el usuario, y además se ayuda de cierta información o conocimiento almacenados en el sistema, normalmente en forma de *léxico*.

2.3.2.1. Mapeo semiautomático

En este tipo de sistemas los términos extraídos de la consulta en lenguaje natural son mapeados a conceptos contenidos en el léxico. De esta forma es posible introducir razonamiento basado en la información semántica almacenada en estos conceptos, lo que permite un análisis más en profundidad de la consulta. En caso de no haber una única alternativa en cuanto a un mapeo término-concepto, el sistema pedirá ayuda al usuario para resolver la ambigüedad a través de un diálogo de clarificación. Como contrapartida, la generación de un léxico que abarque un cierto número de dominios de conocimiento es muy costosa. Ejemplos recientes de este tipo de sistemas son FREyA [Damljanović, 2011] y el propio sistema desarrollado en este trabajo.

2.3.2.2. Mapeo automático

Similares a los anteriores, se diferencian de éstos en que son capaces de resolver automáticamente, sin involucrar al usuario, aquellas ambigüedades que puedan surgir durante el mapeo, o bien introducen técnicas para evitar estos casos. Sistemas como PRECISE [Popescu et al., 2004] restringen el conjunto de preguntas soportadas con el objetivo de conseguir una relación 1:1 en el proceso de mapeo, evitando así las ambigüedades, pero reduciendo a cambio el conjunto de preguntas soportadas. Otros, como por ejemplo PANTO [Wang et al., 2007], tratan de resolver las ambigüedades automáticamente mediante métodos avanzados de procesamiento del lenguaje, el uso de diccionarios, léxicos específicos de dominios de conocimiento, y métodos de emparejamiento.

Capítulo 3

Análisis de antecedentes y alternativas

En este capítulo veremos algunos ejemplos relevantes de sistemas con prestaciones similares a las del desarrollado en este proyecto. Pero antes de empezar conviene introducir el concepto de personalización de una ILN a datos estructurados.

La *personalización* de una ILN a datos estructurados consiste en poblar el léxico de la misma con *lexicalizaciones*[1] de los conceptos y estructuras que aparecen en el esquema de la base de datos o base de conocimiento sobre la que se desee trabajar. Por ejemplo, si queremos personalizar nuestra ILN para trabajar con una ontología sobre organizaciones, debemos introducir conceptos en el léxico tales como "organización", "tamaño de organización", "localización", etc. Este proceso se realiza normalmente con el objetivo de *portar* el sistema a un nuevo dominio de conocimiento. La forma de llevarlo a cabo difiere según el sistema, aunque normalmente nos encontraremos con dos maneras de poblar el léxico: manual o automática. La manual requiere un mayor esfuerzo por parte del experto en el dominio de conocimiento al que se desea portar el sistema, aunque suele obtener mejores resultados. Por el otro lado, la automática resulta más rápida y cómoda, pero puede dar lugar a un léxico insuficiente. Además, puede haber casos en los que la estructura de la base de datos/conocimiento no sea accesible en su totalidad y/o por medios automatizados, como es nuestro caso debido a las restricciones de acceso impuestas por la empresa.

Cabe destacar que la gran mayoría de ILNs han sido desarrolladas para el inglés, tal y como sucede con los sistemas descritos a continuación, mientras que nuestro sistema, aunque preparado para su extensión a otros idiomas, opera originalmente sobre el español, mucho

[1] El proceso de *lexicalización* es aquel en el que se transladan conceptos abstractos a palabras en algún idioma.

más complejo tanto a nivel léxico como morfológico y sintáctico y, consecuentemente, más difícil de procesar, a lo que hay que unir los problemas motivados por la escasez de recursos y herramientas de PLN libremente disponibles para este idioma [Vilares et al., 2008].

3.1. ORAKEL

ORAKEL [Cimiano et al., 2008] es una ILN a ontologías que soporta preguntas sobre hechos que empiecen por una *WH-word* o interrogativo como "who", "where", "what", etc. Las respuestas a estas preguntas son *hechos*, tal y como aparecen recogidos en la base de conocimiento, y en ningún caso una explicación elaborada (ver Apartado 2.1.2). La ventaja de este sistema con respecto a otros similares es el soporte para preguntas en las que aparecen cuantificaciones, conjunciones o negaciones [Damljanović, 2011].

Al igual que el resto de sistemas de este tipo, ORAKEL maneja un léxico de conceptos sobre los dominios de conocimiento sobre los que trabaja. En el caso de este sistema concreto, el léxico se encuentra dividido en dos partes:

- *Léxico genérico.* Compartido por todos los dominios de conocimiento y donde se almacenan palabras como las *WH-word*s.
- *Léxico de dominio específico.* Formado por los conceptos propios de un determinado dominio de conocimiento. Se divide a su vez en dos partes:
  - *Léxico de la ontología.* Extraído de forma automática de la propia ontología sobre la que se desea trabajar.
  - *Léxico para el mapeo de relaciones de la ontología a palabras.* Formado por los mapeos de estructuras linguísticas tales como verbos con sus argumentos, nombres con sus argumentos, etc. Este léxico es creado manualmente por un experto en sucesivos ciclos incrementales tras cada una de las sesiones de interacción del usuario con el sistema, mejorando así la cobertura del léxico.

Un punto débil de esta aproximación es que no siempre existen lexicalizaciones comprensibles y útiles de las relaciones entre conceptos establecidas en la ontología, a pesar de que el sistema así lo considere. Por otra parte, la interacción del usuario con el sistema está limitada a la formulación de la consulta y posterior visualización de los resultados, sin posibilidad de tener control alguno sobre el proceso de traducción de su consulta en lenguaje natural a lenguaje formal.

3.2. QuestIO

Question-based Interface to Ontologies (QuestIO) [Damljanović, 2011] es otra ILN a ontologías capaz de derivar automáticamente la respuesta a preguntas escritas tanto en lenguaje natural como expresadas en forma de un conjunto de palabras clave. Esto se consigue traduciendo la consulta de entrada a SeRQL/SPARQL[2] y devolviendo la respuesta al usuario después de ejecutar la consulta en lenguaje formal contra la ontología dada.

La construcción del léxico se realiza de manera automática. Como consecuencia, el rendimiento del sistema es directamente proporcional a la calidad de las descripciones formales de los conceptos de la base de conocimiento: cuantos más metadatos contengan los recursos semánticos, mayor será la riqueza del léxico generado y, consecuentemente, mejor será el rendimiento del sistema. El procesamiento de la consulta se divide en dos fases:

1. *Interpretación de la consulta.* En esta fase se ejecutan dos acciones sobre cada lema correspondiente a las palabras que conforman la consulta. Operar sobre los lemas en lugar de sobre las palabras originales permite tratar de manera indiferente todas las flexiones morfológicas de una palabra dada.
    - *Búsqueda de conceptos de la ontología.* Aquellos lemas que se correspondan con algún concepto de la ontología se guardarán como *conceptos clave.*
    - *Acumulación del contexto.* El resto de lemas que no se correspondan con ningún concepto de la ontología serán guardados igualmente, pero aparte, puesto que podrían ser útiles más tarde de cara a la construcción de la consulta formal.
2. *Análisis de la consulta.* Cuando todos los datos relevantes han sido recogidos, se procede a realizar las siguientes acciones sobre ellos:
    - *Filtrado de los conceptos clave.* En el lenguaje humano es posible usar la misma expresión en contextos distintos de tal forma que expresen significados totalmente diferentes. De esta forma, es posible que un determinado lema de la consulta se haya hecho corresponder con varios conceptos de la ontología. Esta situación de ambigüedad suele resolverse comprobando el grado de correspondencia del lema con cada uno de los conceptos alternativos, que suele ser distinto, y dar prioridad al concepto que mejor se corresponda.
    - *Identificación de todas las posibles relaciones entre los conceptos clave.* En este paso se tratan de identificar aquellas relaciones definidas en la ontología existentes entre los conceptos clave. Normalmente son recuperadas mediante un proceso de razonamiento basado en la ontología.

[2]Los lenguajes formales de consulta para ontologías más comunes.

- *Clasificación de las relaciones.* Una vez que se tienen todas las relaciones entre los conceptos clave, se procede a puntuar cada una de ellas de acuerdo con alguna métrica establecida, de modo que podamos establecer una clasificación de relaciones.
- *Generación de la consulta SeRQL.* Cuando todas las relaciones hayan sido puntuadas y clasificadas, se podrá generar la consulta formal en SeRQL utilizando todos los datos obtenidos en las fases anteriores.

Las prestaciones más destacables de QuestIO son la flexiblidad del lenguaje soportado, tanto lenguaje natural como palabras clave, y la portabilidad sin necesidad de personalización.

3.3. FREyA

Como evolución de QuestIO aparece Feedback, Refinement and Extended Vocabulary Aggregation (FREyA) [Damljanović, 2011], sistema con el que se trata de dar un paso adelante en términos de usabilidad.

De acuerdo con [Brooke, 1996], la usabilidad puede ser definida como la cualidad general de lo apropiado que es un determinado artefacto para un determinado propósito. Los métodos empleados para aumentar la usabilidad en FREyA son los siguientes:

- *Feedback.* Proporcionar al usuario más información acerca del tratamiento de su consulta y de los resultados obtenidos puede ayudarlo a conocer mejor las prestaciones y posiblidades del sistema. Esto permite evitar la mayoría de las situaciones en las que el usuario se siente frustrado por no saber por qué el sistema le está ofreciendo una respuesta insatisfactoria a su pregunta.
- *Diálogos de clarificación.* Hacen posible solucionar situaciones en las que el sistema no es capaz de determinar de manera automática cómo debe actuar. De esta forma, el usuario puede guiar al sistema de tal modo que obtenga finalmente la respuesta que deseaba.
- *Aprendizaje basado en las elecciones del usuario.* Aprovechando las elecciones tomadas por el usuario en diálogos de clarificación previos, el sistema estará en disposición de aprender a solucionar en un futuro, de forma automática, aquellas situaciones que habían dado lugar a dichos diálogos.

FREyA posee un flujo de trabajo similar al de QuestIO, al que añade nuevas etapas para aumentar el rendimiento:

- *Análisis sintáctico y detección de posibles conceptos de la ontología.* En esta etapa se usan reglas heurísticas sobre los resultados del análisis sintáctico de la consulta para identificar los llamados *posibles conceptos de la ontología.* Esta etapa se ejecuta de forma paralela a la *búsqueda de conceptos de la ontología.*

- *Consolidación.* Una vez obtenidos los denominados *conceptos* y los *posibles conceptos* de la ontología, el algoritmo de consolidación tratará de combinar ambos tipos de elementos de modo que al final se quede únicamente con los conceptos verdaderamente presentes en la ontología. Si no es capaz de completar con éxito este proceso de forma automática, el sistema pedirá ayuda al usuario a través de un diálogo de clarificación. Los diálogos de clarificación pueden ser a su vez de dos tipos, cada uno de los cuales busca solucionar uno de los dos posibles problemas a los que el algoritmo de consolidación se puede enfrentar:

  - *Diálogo de desambiguación.* Ocurre cuando, como en el caso de QuestIO, a un determinado término de la consulta le corresponden varios conceptos de la ontología. El usuario deberá elegir en el diálogo uno de estos conceptos.
  - *Diálogo de mapeo.* Ocurre cuando un posible concepto de la ontología queda libre después de la etapa de consolidación, sin corresponderse con ningún concepto. El sistema tratará de encontrar conceptos relacionados con éste, alguno de los cuales será el que se corresponde realmente con el posible concepto, y se los mostrará al usuario para que elija el que considere correcto.

Capítulo 4

Estudio de viabilidad

Dado que buena parte del sistema desarrollado en este proyecto pasará a manos de la empresa Classora Technologies, se consideró adecuado emplear las mismas herramientas de desarrollo que se usan en dicha empresa para la implementación de nuestro sistema. Esto reporta dos ventajas importantes con respecto al uso de otras herramientas alternativas: por una parte se consigue una rápida integración del sistema desarrollado en el seno de la empresa y, por la otra, se aprovecha el soporte que el personal de la misma pueda ofrecer sobre la utilización de estas herramientas.

4.1. Frameworks de programación

Nuestro sistema se sustenta sobre la plataforma de programación Java, concretamente sobre la Java 2 Platform, Enterprise Edition (J2EE),[1] el cual permite estructurar el sistema según una arquitectura de varias capas. Desde un punto de vista global, las capas a implementar serán las definidas por el patrón de diseño Modelo Vista Controlador (MVC) [Gamma et al., 1995]. Cada una de estas capas se ha desarrollado apoyándose en uno o más *frameworks* de programación, tal y como se describe a continuación:

- *Capa modelo*. En esta capa se aglutina toda la lógica de negocio de la aplicación. En este caso, se trata justamente de lo que en apartados anteriores denominábamos *núcleo* del sistema. Se divide a su vez en dos subcapas:
  - *Persistencia*. Para la implementación de la persistencia de datos se utilizó una base de datos relacional SQL gestionada por MySQL.[2] Para su integración con

[1] http://www.oracle.com/technetwork/java/javaee/overview/index.html
[2] https://www.mysql.com

J2EE se hizo uso del Mapeo Objeto Relacional (*Object–Relational Mapping*) (ORM) proporcionado por el framework Hibernate.[3]

- *Servicio*. En esta capa se implementa el patrón de diseño Inversión de Control (*Inversion of Control*) (IoC) a través del framework Spring,[4] que además provee del aspecto de *transaccionalidad* a los servicios.

Para la implementación de las pruebas automatizadas de unidad e integración sobre estas capas, se ha empleado JUnit.[5]

- *Capa web*. Dentro de esta capa se engloban las otras dos capas especificadas en el patrón MVC: capa vista y capa controlador. La implementación del patrón en esta capa se consiguió mediante la utilización del framework Apache Struts 2.[6] Asimismo, la integración con la capa modelo se lleva a cabo mediante el plugin de Struts `struts2-spring-plugin`, que permite el uso de los servicios definidos mediante Spring, y almacenados en su contenedor, a través de Struts.

4.2. Herramientas de PLN

Parte de la funcionalidad de PLN necesaria para la lógica de negocio de la capa modelo pudo obtenerse mediante el uso de librerías externas de PLN. Llegados a este punto debemos llamar de nuevo la atención sobre el hecho de que nuestro sistema, aunque preparado para su migración a otros idiomas, ha sido desarrollado para español, lo que de por sí supone una novedad, ya que este tipo de sistemas suelen centrarse en el inglés. Por otra parte el empleo del español supone una dificultad adicional debido tanto a su mayor complejidad lingüística respecto al inglés como a la escasez de recursos y herramientas de PLN libremente disponibles para este idioma [Vilares et al., 2008]. La funcionalidad buscada con estas herramientas es la siguiente:

- *Tokenizer*. División de la consulta en sus términos constituyentes o *tokens*, normalmente palabras y signos de puntuación [Dale et al., 2000, Ch. 2].
- *Corrección ortográfica*. Eliminación de los errores ortográficos presentes en la consulta [Jurafsky and Martin, 2009, Ch. 3].
- *Lematización*. Extracción de los lemas de cada uno de los términos de la consulta, es decir, obtención de su forma canónica (lo que sería su correspondiente entrada en un diccionario de la lengua).

[3] http://www.hibernate.org
[4] http://www.springsource.org
[5] http://junit.org
[6] https://struts.apache.org

- *Etiquetación morfosintáctica (PoS tagging).* Obtención de las etiquetas morfosintácticas de cada uno de los términos de la consulta [Jurafsky and Martin, 2009, Ch. 5] [Dale et al., 2000, Ch. 17]. En otras palabras, identificar su categoría gramatical (si se trata de un sustantivo, verbo, adjetivo, etc.), así como su flexión (género, número, tiempo, modo, etc.).

- *Reconocimiento de Entidades con Nombre (REN).* Detección de las entidades mencionadas en las consultas, que pueden ser de diversos tipos (nombres de lugares, personas, organizaciones, etc) [Jurafsky and Martin, 2009, Ch. 22].

La función de corrección ortográfica fue integrada gracias al servicio web que pone a disposición Google para su corrector ortográfico, Google Spell Checker, a través del API Java `google-api-spelling-java`.[7] Sin embargo, la integración del resto de librerías de PLN fue un proceso complejo, habiendo casos en los que fue necesario probar varias alternativas hasta encontrar la más adecuada. A continuación se enumeran las distintas herramientas de PLN consideradas como candidatas para cubrir el resto de funcionalidades requeridas por el sistema:

1. *Freeling.*[8] Se trata de una librería escrita en C, muy completa, que provee de servicios para el análisis del lenguaje natural, y distribuida bajo la licencia GNU General Public License (GPL) [GNU Project, 2013]. Entre los idiomas soportados se encuentran el gallego, el español y el inglés. Fue la librería que más problemas ocasionó. Se empezó con graves dificultades para su compilación y utilización bajo el entorno de trabajo (Mac OS X), siendo el soporte además muy limitado para este entorno. Ésta fue la principal razón por la que se decidió cambiar la estructura del entorno de trabajo, tal y como se explica en la Sección 4.4. Finalmente, aún a pesar de este cambio el empleo de esta herramienta tuvo que desestimarse ya que ofrecía un comportamiento errático al recibir llamadas desde Java a través de un *wrapper*.

2. *Apache OpenNLP.*[9] Conjunto de herramientas PLN escritas en Java y basadas en aprendizaje automático, pensadas para utilizar encadenadas mediante *pipelines*. Como todo proyecto o software producido por la fundación Apache, se encuentra distribuido bajo la licencia Apache License [ASF, 2013]. Al tratarse de una librería implementada en Java, su integración con el sistema no supuso problemas. Sin embargo, dado que cada uno de los componentes que la conforman requieren de un modelo de datos para funcionar, nos encontramos con el inconveniente de tener que

[7] https://code.google.com/p/google-api-spelling-java
[8] http://nlp.lsi.upc.edu/freeling
[9] https://opennlp.apache.org

encontrar modelos para estos componentes para el caso del español. Una vez encontrado el modelo para el etiquetador morfosintáctico, se comprobó que no existía documentación sobre el conjunto de etiquetas empleadas por dicho modelo. Esto resultó ser un problema grave, puesto que de pretender usar este componente, se perdería mucho tiempo averiguando con qué categoría morfosintáctica se corresponde cada una de las etiquetas definidas, lo cual motivó que se buscasen alternativas. Otros componentes de esta librería, como el *tokenizer* o el sistema de REN, sí se consideraron útiles, y fueron integrados.

3. *Treetagger*.[10] Herramienta que permite anotar un texto en lenguaje natural con etiquetas morfosintácticas y extraer los lemas de sus términos. Inicialmente este componente fue introducido en el sistema para suplir una carencia de OpenNLP; el lematizador. Si bien se trata de una herramienta escrita en C, su integración con Java mediante un wrapper (`tt4j`)[11], al contrario que en el caso de Freeling, no supuso ningún problema. Más adelante, cuando se desestimó el uso del etiquetador de OpenNLP por las razones anteriormente expuestas, se introdujo esta funcionalidad desde Treetagger, ya que éste sí posee documentación sobre el conjunto de etiquetas empleadas. La licencia bajo la que se distribuye únicamente permite su uso para propósitos educativos, de investigación y evaluación.

Cabe destacar que cualquiera de estas herramientas podría ser intercambiada fácilmente por otras si más adelante el sistema se comercializase y las licencias pudieran resultar un impedimento para ello.

4.3. Herramientas de Aprendizaje Automático

La mayoría de los sistemas de BR hacen uso de un *clasificador* para identificar el tipo de pregunta formulada. Esta clasificación suele obtenerse a través de aproximaciones basadas en Aprendizaje Automático (AA), las cuales requieren de grandes cantidades de datos de entrenamiento durante la fase de aprendizaje para operar correctamente [Damljanović, 2011].

El sistema desarrollado en este proyecto emplea esta aproximación mediante el uso del *framework* de AA *Weka*,[12] el cual provee de algoritmos de clasificación (además de muchos otros de diversos tipos: preprocesado, regresión, *clustering*, etc.) y está distribuido bajo la licencia GPL [GNU Project, 2013].

[10] http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger
[11] https://code.google.com/p/tt4j
[12] http://www.cs.waikato.ac.nz/ml/weka/

4.4. Entorno de desarrollo

En un principio, el entorno de desarrollo estaba formado por el Sistema Operativo (SO) Mac OS X, el Entorno de Desarrollo Integrado (*Integrated Development Environment*) (IDE) NetBeans,[13] la herramienta gráfica de gestión de bases de datos MySQLWorkbench,[14] el gestor de proyectos Apache Maven[15] y el Sistema de Control de Versiones (SCV) Apache Subversion.[16] Tras los problemas sufridos durante la integración de la librería Freeling ya descritos en la Sección 4.2, se decidió cambiar la estructura de este entorno de desarrollo. La principal premisa de este cambio era flexibilizar todo lo posible el entorno, haciéndolo lo más independiente posible del SO subyacente, que en este caso era el componente que dificultaba la integración de Freeling. Si bien se podría pensar en cambiar el SO de la máquina como solución rápida, el requisito de flexibilidad conlleva recurrir a la otra solución inmediata: la *virtualización* del SO. De esta forma, el entorno global de desarrollo se divide ahora en otros dos entornos, ambos ejecutados sobre un SO anfitrión Mac OS X:

- *Entorno de programación/compilación.* Este entorno se ejecuta directamente en el SO anfitrión, y está formado por las herramientas Netbeans, MySQLWorkbench, Maven y Subversion. Desde aquí se gestiona, se escribe y se compila el código directamente a través del IDE, mientras que el diseño e implementación de la base de datos se realiza a través de MySQLWorkbench.

- *Entorno de ejecución y pruebas.* Este entorno se ejecuta en un SO invitado Ubuntu Server 12.04,[17] dentro de una Máquina Virtual (VM) Oracle VirtualBox.[18] Se decidió añadir, además, una capa de gestión sobre esta virtualización a través del *middleware* Vagrant.[19] El uso de Vagrant permite una rápida replicación del entorno de ejecución y pruebas en cualquier otra máquina, incrementando el nivel de flexibilidad del entorno global. Para la elección del SO anfitrión se tuvo en cuenta el SO que se utiliza en el entorno de producción de Classora Technologies, lo que más adelante facilitará el despliegue final de la aplicación. En este entorno se ejecutarán las pruebas y la propia aplicación a través de Maven, empleando Jetty[20] como servidor web en los casos necesarios.

[13] https://netbeans.org
[14] https://www.mysql.com/products/workbench
[15] https://maven.apache.org
[16] https://subversion.apache.org
[17] http://releases.ubuntu.com/precise
[18] https://www.virtualbox.org
[19] http://www.vagrantup.com
[20] http://www.eclipse.org/jetty

Todo el código y configuración contenidos en el proyecto Maven son compartidos entre ambos SO, anfitrión e invitado, a través de la funcionalidad de carpetas compartidas provista por VirtualBox. Es de esta manera como se consigue comunicar ambos entornos, conformando el entorno de desarrollo global. Cabe destacar también que todas las herramientas empleadas durante el desarrollo de la aplicación son multiplataforma.

Parte III

Construyendo Interfaces de Lenguaje Natural

Capítulo 5

Metodología

En este capítulo se expone el proceso de desarrollo de nuestra ILN, dividido en las fases habituales de un proyecto software: análisis, diseño, implementación y pruebas. Además, por tratarse de una aplicación web en la que se aplica el patrón de diseño MVC (véase Sección 4.1), la etapa de diseño se encuentra dividida en dos, una para la capa modelo y la otra para la capa web. El resto de etapas se describen desde un punto de vista global del sistema, particularizando los aspectos de una u otra capa según sea necesario. En todo caso, se hará uso de diagramas UML para una correcta especificación del análisis y diseño, manteniendo su nivel de detalle al requerido para la correcta explicación de cada apartado. En caso de que el lector necesitara acceso a diagramas más detallados, éstos pueden encontrarse en la carpeta de nombre "`uml`" en el CD adjunto a la presente memoria.

5.1. Análisis

En esta primera fase se parte del enunciado del proyecto, provisto por Classora Technologies, del cual se tratarán de extraer los aspectos fundamentales del sistema a desarrollar. Los resultados de esta fase vienen dados en su mayor parte en forma de diagramas UML, en base a los cuales se divide esta sección: casos de uso, actividades y componentes.

5.1.1. Casos de uso

El primer paso en el proceso de desarrollo es, partiendo de los requisitos especificados en el enunciado del proyecto, comprender la forma en la que el sistema va a ser utilizado por parte del usuario, lo que en ingeniería del software suele denominarse *casos de uso*.

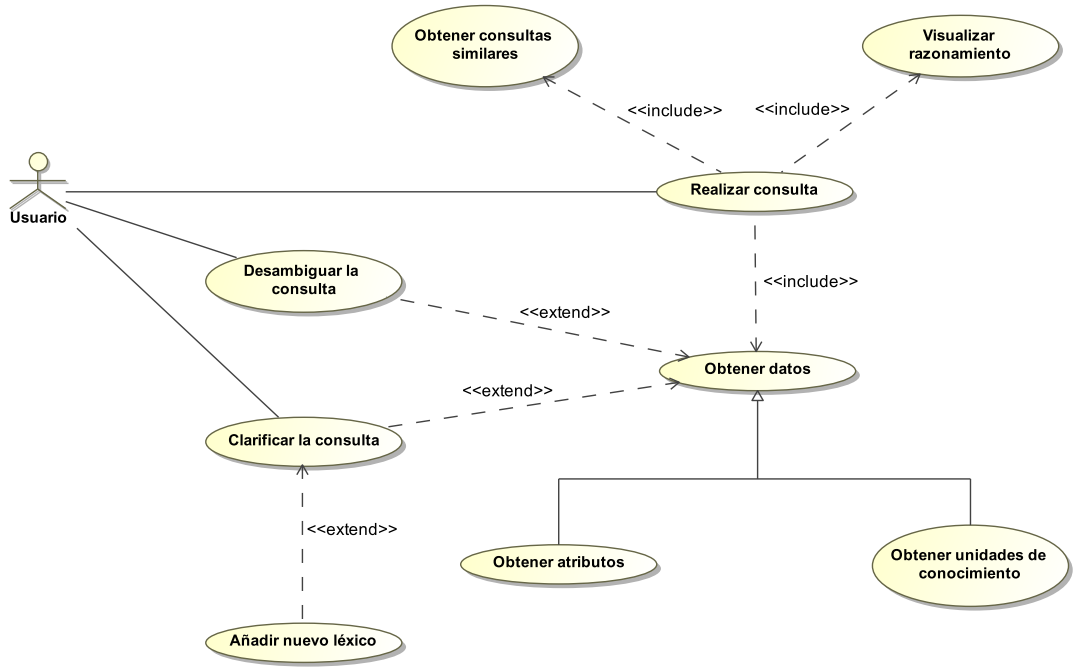


Figura 5.1: Diagrama de casos de uso.

En la Figura 5.1 se muestra un diagrama de casos de uso. Tres son los casos base, los cuales están directamente relacionados con el usuario:

- *Realizar consulta*. Se trata del caso más básico de todos, y se corresponde con la forma más fundamental de interacción entre el usuario y el sistema: la formulación de una pregunta. Esto conlleva a su vez la *obtención de datos* de la base de conocimiento, así como el enriquecimiento de los resultados mediante la *obtención de consultas similares* a la realizada y la *visualización del razonamiento* seguido por el sistema. En cuanto a la naturaleza de los datos a obtener, pueden distinguirse dos casos concretos de este caso de uso en base al tipo de la pregunta, descrito en la Sección 1.3: *obtener atributos* (para preguntas de Tipo 1) y *obtener unidades de conocimiento* (para preguntas de Tipo 2).

- *Desambiguar la consulta*. Tras el envío de la consulta al sistema, y como extensión a la funcionalidad base de obtener datos, se podría requerir la intervención del usuario para desambiguar algún término de la consulta cuando existen varias interpretaciones posibles para el mismo.

- *Clarificar la consulta*. En un caso similar al anterior, el usuario podría ser requerido para aclarar el concepto al que se refiere con una determinada palabra de su consulta.

Este proceso, repetido un cierto número de veces para la misma palabra, dará lugar a la inserción de ésta en el léxico del sistema (*añadir nuevo léxico*).

5.1.2. Actividades

Una vez que sabemos *qué* funcionalidad ha de soportar nuestro sistema, debemos realizar una primera aproximación al *cómo* ha de implementarse dicha funcionalidad. Para ello, nos ayudaremos de los diagramas de actividades que ofrece el estándar UML.

En la Figura 5.2 se puede observar el flujo de trabajo de nuestro sistema, del que podemos extraer dos grandes grupos de actividades: aquellas que intervienen desde la recepción de la consulta en lenguaje natural hasta la generación de la consulta en lenguaje formal, y aquellas otras que comienzan en este punto y continúan hasta la presentación de los resultados al usuario. Con esta primera división acabamos de separar la lógica de la capa modelo de la lógica de la capa web.

Del grupo de actividades de la capa modelo se pueden extraer los subsistemas que conforman el núcleo del sistema completo. De este modo, se pueden distinguir cuatro fases bien diferenciadas a partir de estas actividades: preprocesado, análisis, interpretación y generación de la consulta en lenguaje formal.

- *Preprocesado.* En esta fase, en la que se incluye la recepción de la consulta en lenguaje natural, se lleva a cabo un procesamiento preliminar de esta de modo que, por ejemplo, se corrijan errores ortográficos. Ha de tratarse de una etapa fácilmente extensible.

- *Análisis.* Se incluye en esta fase toda la funcionalidad de PLN provista por herramientas externas. En ella se realiza la división de la consulta en *tokens*, así como la etiquetación morfosintáctica, la lematización de los mismos y la detección de fechas.

- *Interpretación.* Se trata de la fase más compleja del núcleo del sistema. En ella se buscan las correspondencias entre los términos de la consulta y los conceptos de la base de conocimiento, y se extraen estructuras de datos relevantes de la consulta, incluida su clasificación,[1] ayudándose para ello de los resultados del análisis de la etapa anterior. Estas estructuras serán la base para la posterior generación de la consulta en lenguaje formal. Además de todo esto, es en esta fase donde puede ser necesaria la interacción con el usuario a través de los diálogos de desambiguación y/o clarificación. En la Figura 5.3 se presenta un diagrama de actividades para esta etapa concreta.

[1] Las consultas se clasifican en los dos tipos vistos en la Sección 1.3: Tipo 1 y Tipo 2.

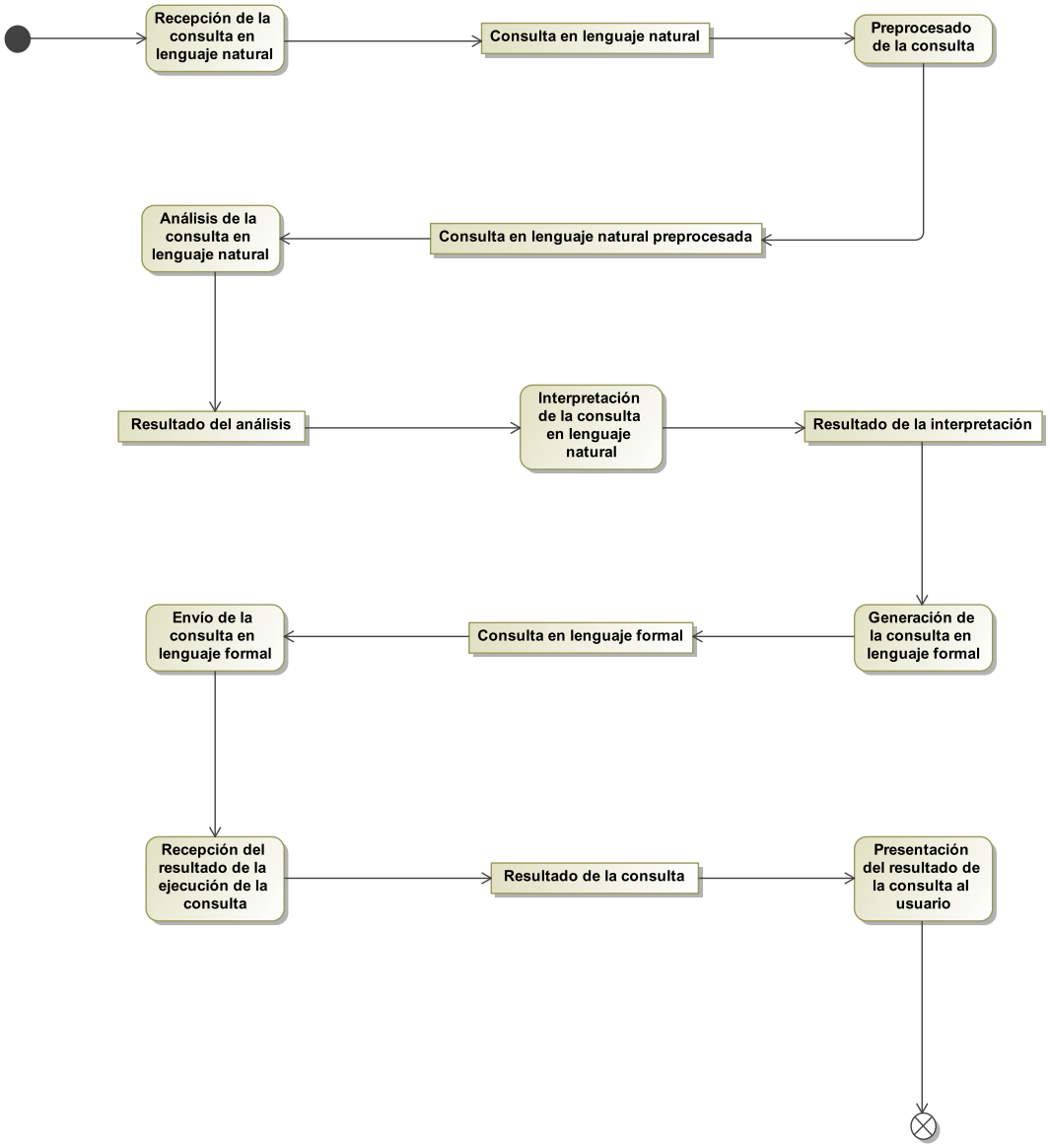


Figura 5.2: Diagrama general de actividades.

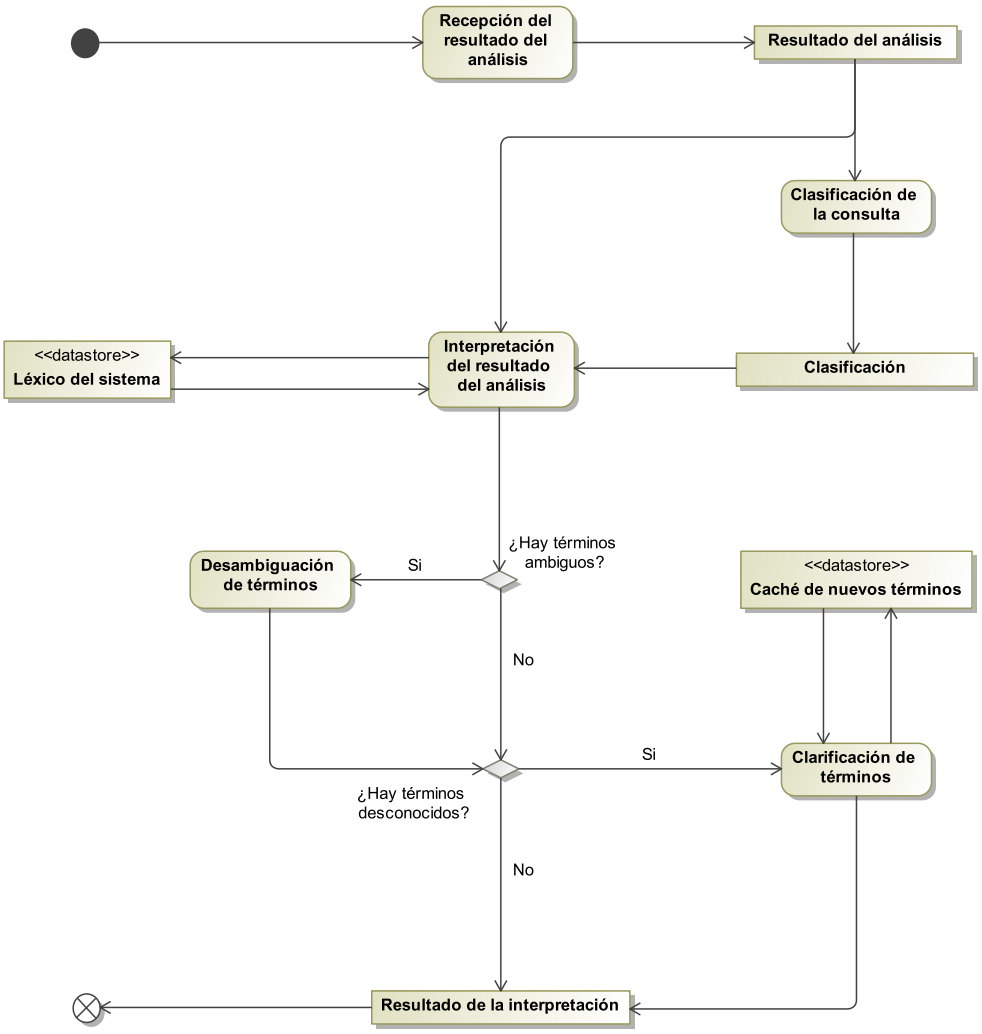


Figura 5.3: Diagrama de actividades del intérprete.

- *Generación de la consulta en lenguaje formal.* Fase final en la que, a partir de los datos obtenidos en la fase de interpretación, se genera la consulta en el lenguaje formal requerido.

5.1.3. Componentes

Del análisis del diagrama general de actividades visto en el apartado anterior se pudieron extraer grupos de actividades que realizan tareas específicas. Cada uno de estos grupos de actividades podrían encapsularse dentro de un componente del sistema, también llamado subsistema, para así conformar entre todos ellos el sistema completo, tal y como se puede ver en la Figura 5.4.

5.2. Diseño de la capa modelo

Dado que el proceso de desarrollo seguido es orientado a objetos, una vez tenemos los resultados del análisis, se intentarán modelar según este paradigma las diferentes clases que conforman cada componente del sistema completo. En esta sección se hará un recorrido por todos los componentes del núcleo, dejando para la siguiente aquellos pertenecientes a la capa web.

Las clases de esta capa se encuentran almacenadas en subpaquetes de `es.udc.pfc.nloki.model` de la siguiente manera:

- Las entidades y sus Data Access Object (DAO) se encuentran en un paquete con el mismo nombre que la propia entidad; de esta forma `Lex` se ubicaría en `es.udc.pfc.nloki.model.lex`.
- Los servicios se encuentran a su vez bajo el paquete `services`, cada uno en un subpaquete del mismo nombre que el propio servicio; de este modo `LexService` se ubicaría en `es.udc.pfc.nloki.model.services.lexservice`.
- Bajo el paquete `translator` se encuentran el resto de clases, que ya no tienen tanto que ver con las entidades del sistema, sino más bien con el proceso de traducción en sí. Cada etapa del proceso tiene sus clases almacenadas en un subpaquete con su mismo nombre (en el caso del intérprete, por ejemplo, se trataría del `es.udc.pfc.nloki.model.translator.interpreter`), las diferentes encapsulaciones de los datos que se obtienen al final de cada etapa se encuentran bajo `query`, y la entidad transitoria `NewLex` se encuentra bajo `newlex`. Por último, la fachada del núcleo del sistema se encuentra en el propio paquete `translator`.

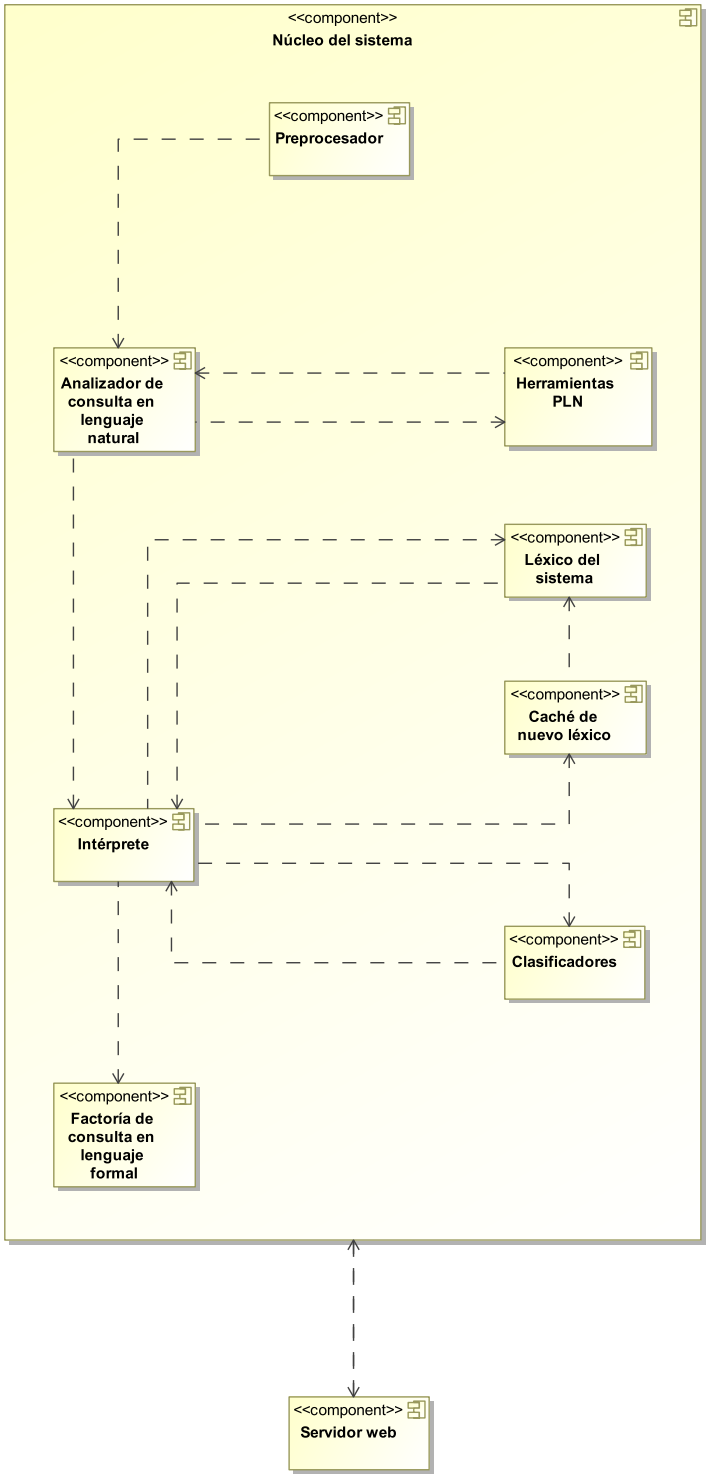


Figura 5.4: Diagrama de componentes del sistema.

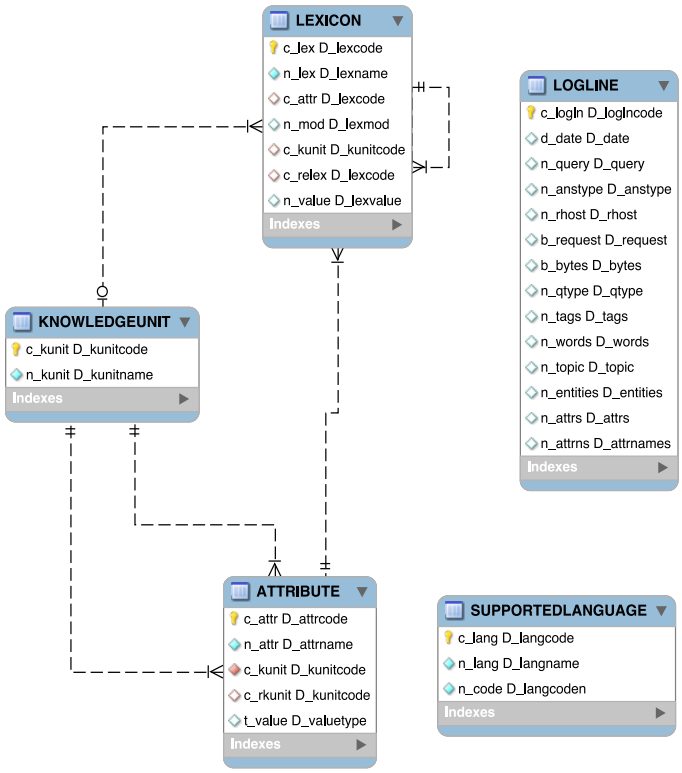


Figura 5.5: Diagrama de entidades de la BD.

Por otra parte, existe un grupo de clases varias que no encajan en ninguno de los paquetes anteriores, puesto que ofrecen funcionalidades de carácter transversal al sistema. Estas clases se almacenan bajo `es.udc.pfc.nloki.util`.

5.2.1. Esquema de la base de datos

Como soporte para las entidades que se verán en el apartado siguiente, se han implementado una serie de tablas en una base de datos MySQL. El diseño del esquema se realizó con la ayuda de la herramienta MySQLWorkbench, mediante la que además se obtuvo el *script* de creación de tablas. En la Figura 5.5 mostramos dicho diseño.

Para la especificación de los tipos de las distintas filas de las tablas, se utilizan *dominios* personalizados, los cuales dotan de una mayor semántica al esquema.

5.2.2. Entidades

Para poder trabajar con CKB necesitamos replicar de alguna manera su estructura de conocimiento, puesto que para el proceso de traducción se necesita cierta información sobre la forma en que se organiza dicho conocimiento. Para ello se toma en cuenta el subconjunto de la estructura expuesta de CKB, que no sólo se tratará de replicar, sino que se introducirán además elementos que probablemente no se encontraran en la original pero que pueden resultar útiles para el tipo de procesamiento que nuestro sistema lleva a cabo. En la Figura 5.6 se pueden observar las clases que conforman las entidades del sistema, con las que trataremos de realizar dicha réplica. Estas entidades pueden ser *persistentes* o *transitorias*, dependiendo de si se han de guardar sus instancias en la base de datos, a través del ORM provisto por Hibernate, o no. El primer grupo está formado por las siguientes clases:

- `KUnit`, abreviatura de *Knowledge Unit*, viene a representar a las unidades de conocimiento de CKB. Dado que estos elementos pueden tener *atributos* asociados se crea además la clase `Attribute`.
- Como soporte para el léxico del sistema tenemos la clase `Lex`, donde se almacenan los conceptos que pueden aparecer en las consultas. Para que éstos puedan resultar útiles durante el procesamiento, se les asocian múltiples metadatos en forma de relaciones con las otras clases descritas hasta ahora o bien simples cadenas de caracteres que actúen de indicadores.
- `LogLine` representa las entradas de un log de ejecución en el que se guardan diversos datos de utilidad sobre el modo de uso de la aplicación por parte de los usuarios, permitiendo así introducir mejoras futuras en el sistema una vez se cuente con suficientes datos acerca del comportamiento de los usuarios (por ejemplo, mejorar el clasificador con nuevas instancias de entrenamiento).
- Para almacenar la información sobre los idiomas de entrada soportados por el sistema usaremos la clase `Language`.

El segundo grupo está formado únicamente por `NewLex`, que es la clase que ofrece el soporte para el aprendizaje de nuevo léxico. Si se demuestra, a través de los diálogos de clarificación, que un determinado término resulta relevante para el procesamiento de un buen número de consultas, pero dicho término no se encuentra actualmente en el léxico del sistema, la instancia correspondiente de `NewLex`, obtenida de los diálogos de clarificación, será convertida en una instancia de `Lex`, con lo que el sistema "aprende" el término.

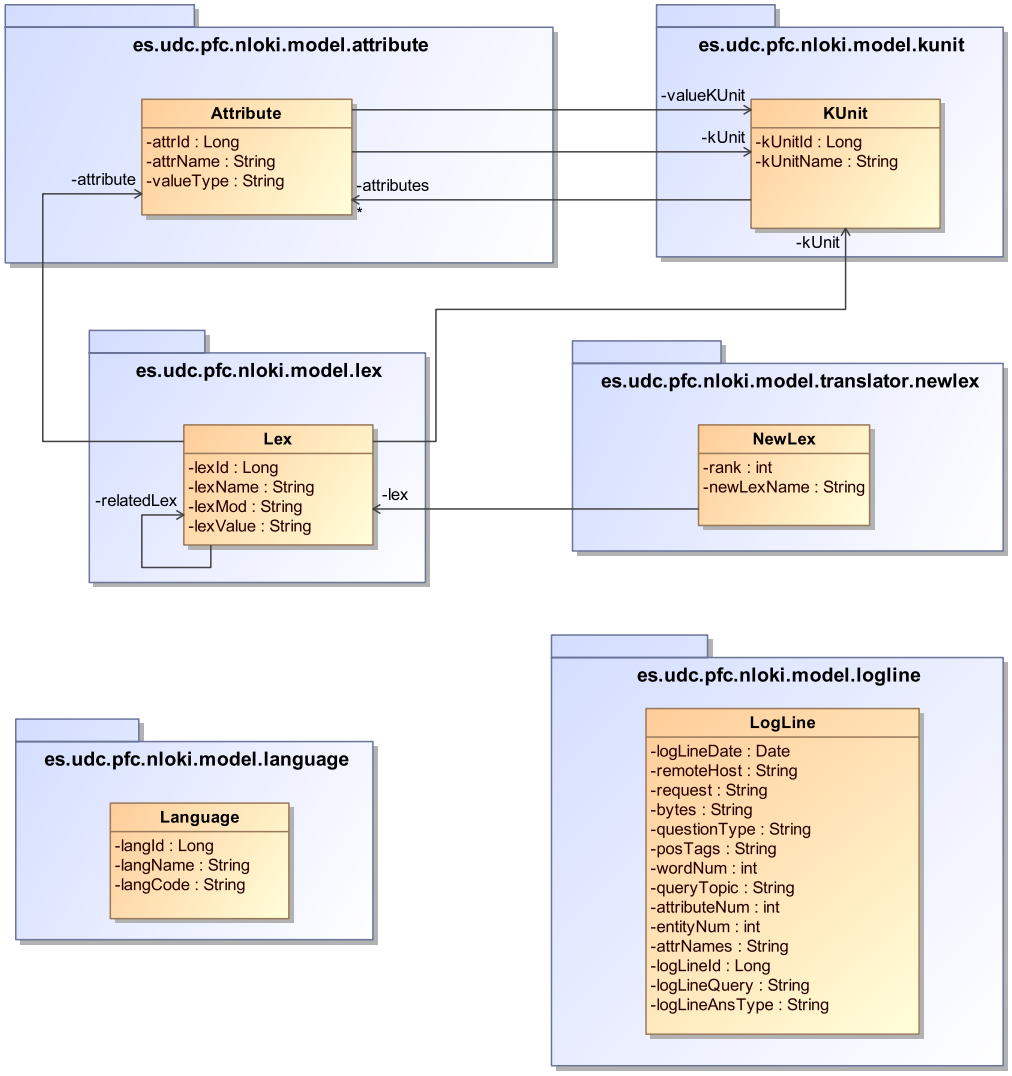


Figura 5.6: Diagrama de clases de las entidades del sistema.

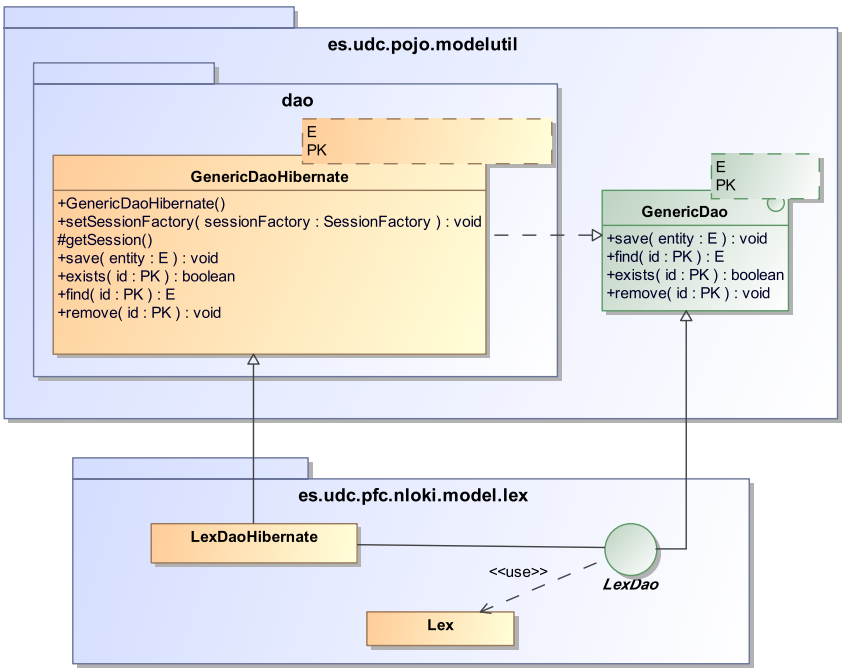


Figura 5.7: Diagrama de clases del DAO para `Lex`.

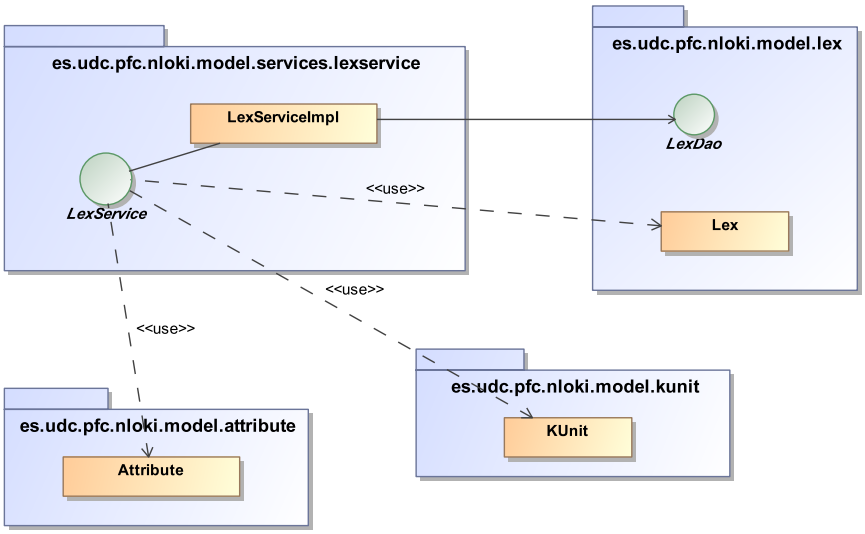


Figura 5.8: Diagrama de clases del servicio para el DAO de `Lex`.

Para emplear las entidades persistentes dentro de otros componentes del sistema, se hace uso del patrón DAO junto a la definición de servicios, que son las clases que proveen de la interfaz para el uso de los métodos definidos por los DAOs. En la Figura 5.7 se pone como ejemplo el DAO correspondiente para la entidad `Lex`, y en la Figura 5.8 se puede observar su servicio, siendo el resto de casos con las demás entidades similares a éste.

Tanto para los servicios de las entidades como para las clases que definen las etapas de procesamiento del sistema, se ha empleado el patrón IoC provisto por Spring, de modo que las implementaciones concretas de cada uno de estos elementos y su uso dentro del propio sistema sean fácilmente intercambiables.

5.2.3. Resultados de cada etapa

Tras la ejecución de cada fase del traductor se generan conjuntos de datos que han de alimentar a la etapa siguiente del proceso. Estos datos serán almacenados de forma conjunta en clases de objetos, lo que permite un tratamiento más eficaz de los mismos. Los nombres de estas clases suelen denotar de qué fase son resultado sus datos: por ejemplo, tras la fase de análisis se da lugar a un objeto de la clase `AnalyzedQuery`. En la Figura 5.9 se pueden ver todas estas clases, además de otras, denominadas *partes*, puesto que suelen estar contenidas en las primeras.

De todas estas clases, debemos destacar el caso de `Predicate`. Con ella se trata de representar una estructura de *triplete*, de la forma `(concepto, operador, valor)`. Esta estructura será la encargada de "condensar" la información encontrada en una consulta, y con la que luego se extrae, en el caso de las preguntas de Tipo 1, el atributo (*foco*) que deseamos obtener sobre una determinada entidad (*tema*); o bien, en el caso de las preguntas de Tipo 2, las condiciones de búsqueda.

5.2.4. Preprocesador

En un apartado anterior se decía que esta etapa debería ser fácilmente extensible. Pues bien, mediante el uso del patrón de diseño *decorador*, se hace posible apilar de forma dinámica procesos con los que tratar la consulta formulada por el usuario. En el supuesto de que no se defina ningún proceso a mayores, siempre se ejecutará el `BasePreprocessor`, que no aplica ninguna clase de tratamiento sobre la consulta. El diagrama de clases de este componente, así como su diagrama de secuencia se pueden ver en las Figuras 5.10 y 5.11, respectivamente.

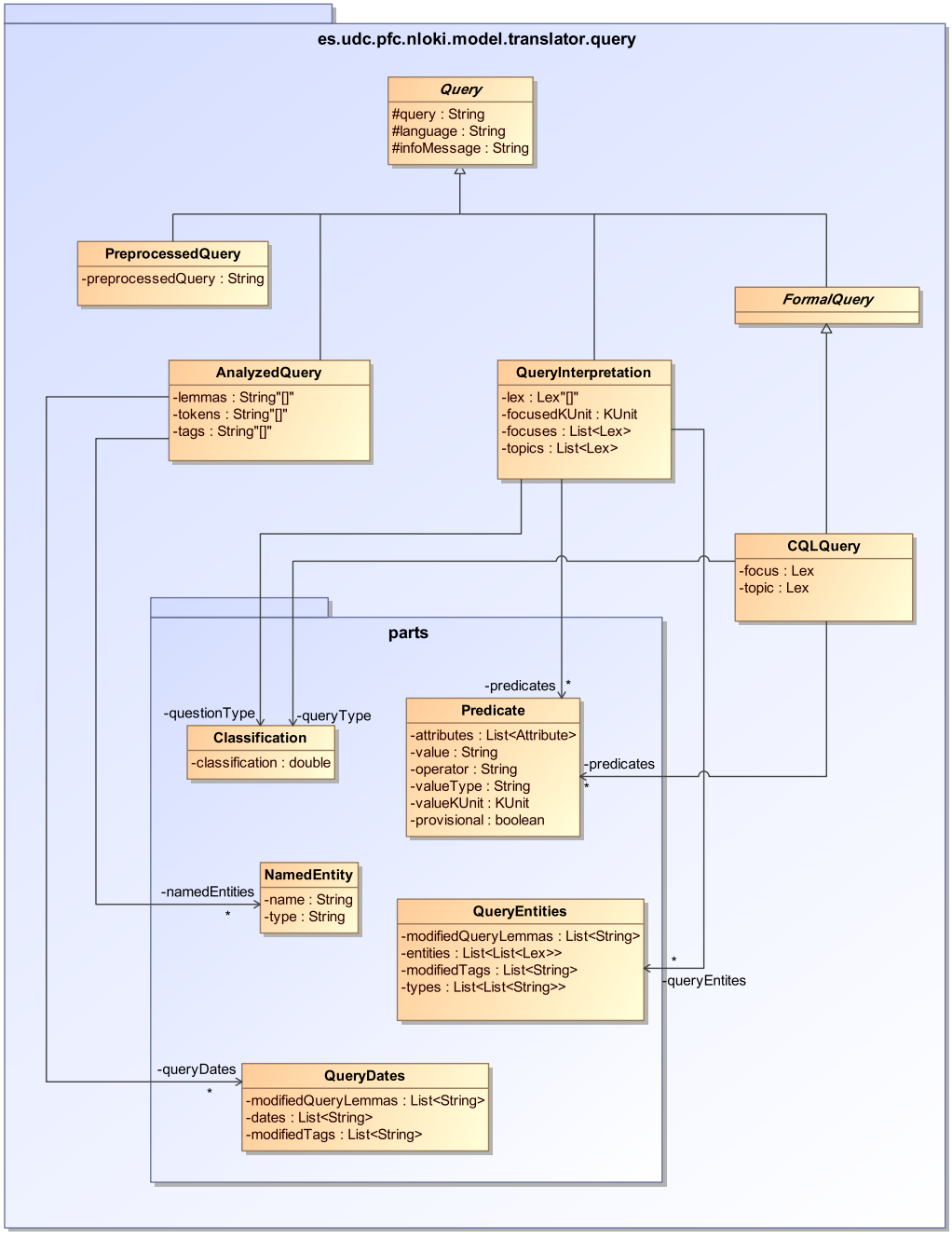


Figura 5.9: Diagrama de clases de los resultados de cada fase.

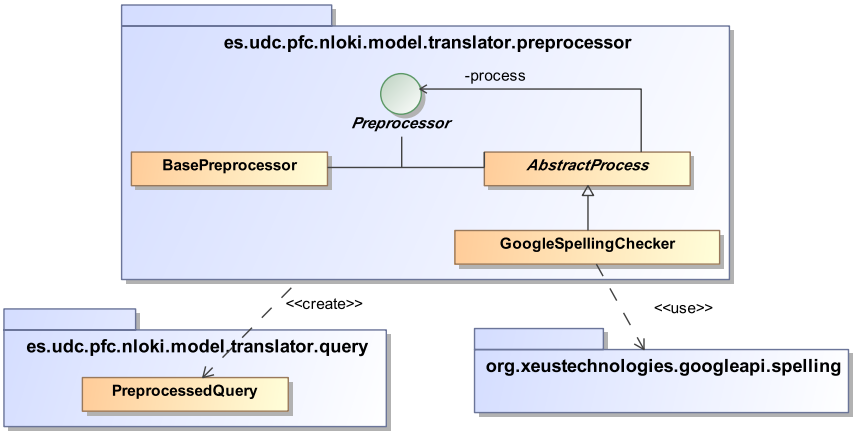


Figura 5.10: Diagrama de clases del preprocesador.

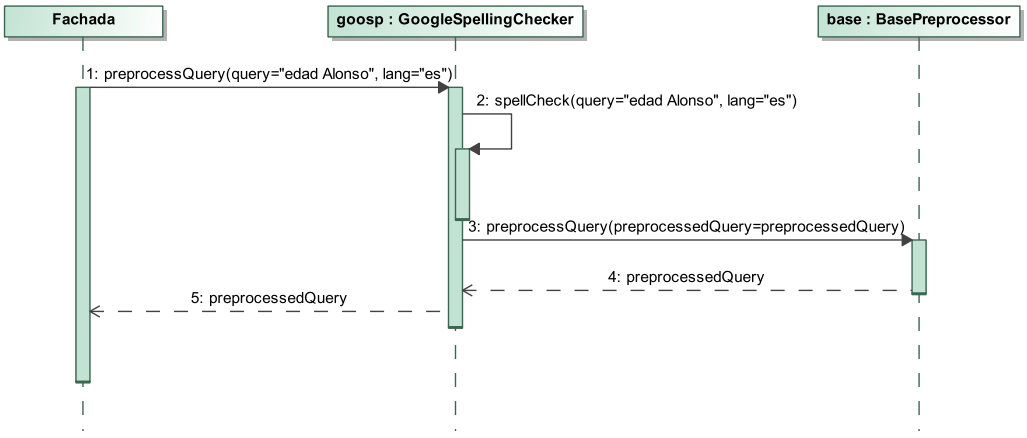


Figura 5.11: Diagrama de secuencia del preprocesador.

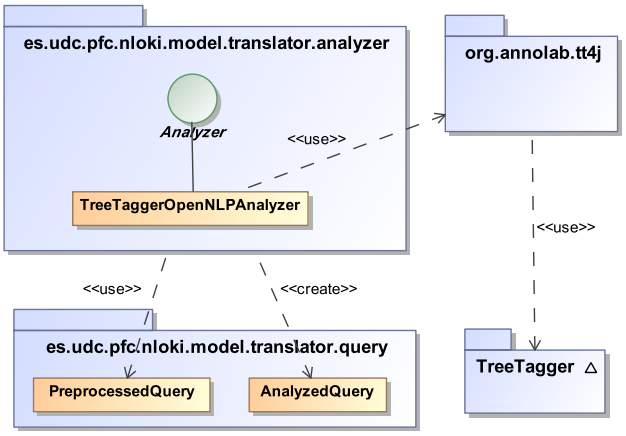


Figura 5.12: Diagrama de clases del analizador.

5.2.5. Analizador

En el analizador se llevan a cabo las tareas de PLN realizadas por medio de herramientas externas, además de la detección de fechas, tal y como se observa en la Figura 5.12.

5.2.6. Intérprete

De los componentes que conforman el sistema, el más complejo de todos es el *intérprete*. A su vez, de los datos que se obtienen durante la etapa de interpretación, los más complejos de conseguir son los predicados (`Predicate`). Sin entrar mucho en detalle, el método para obtener estas estructuras pasa por "rellenar" los campos de un predicado con los conceptos que aparecen en la consulta a medida que avanzamos de izquierda a derecha por sus términos. De encontrarnos con algún término separador (condición indicada por la interpretación de su etiqueta morfosintáctica), se comprueba si el predicado actual está completo: de estarlo, se guarda en la lista de predicados encontrados, en otro caso, se continúa con la formación del predicado actual. Una vez terminada la búsqueda de predicados, se realizarán una serie de comprobaciones de coherencia que permitan descartar instancias de tipos incompatibles del estilo "edad = Galicia".

El comportamiento expuesto, sin embargo, viene dado por una de las posibles implementaciones de *estrategia* de interpretación (de hecho actualmente la única). Y es que el diseño de este componente del sistema se basa en el patrón *estrategia*, que en nuestro caso permite la implementación de diversas formas de obtener los datos de interpretación de una consulta. Así, este patrón fue introducido con objeto de permitir la creación de estrategias para cada uno de los idiomas de entrada a soportar por el sistema, además de

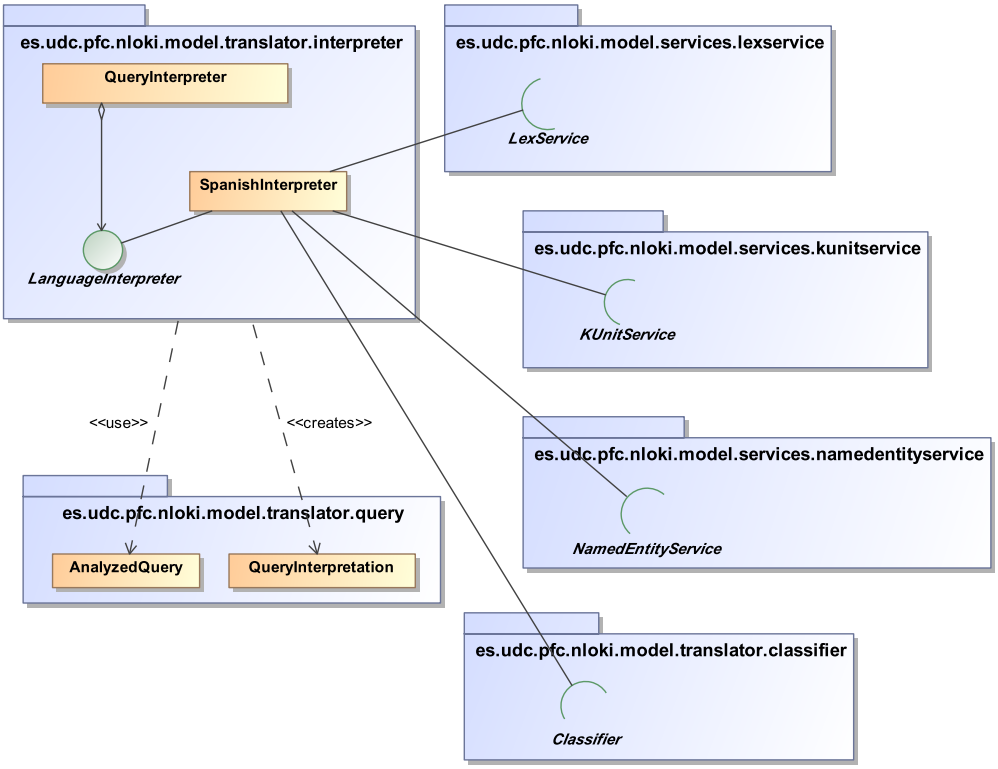


Figura 5.13: Diagrama de clases del intérprete.

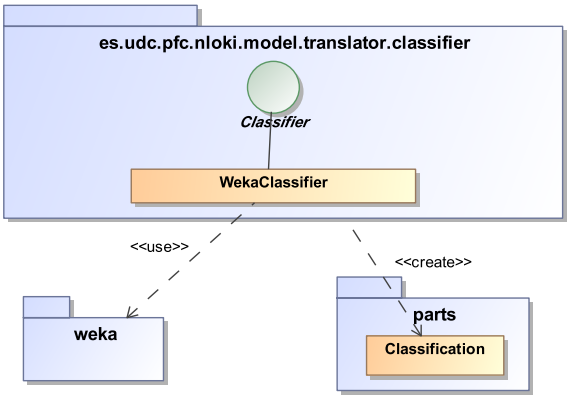


Figura 5.14: Diagrama de clases del clasificador.

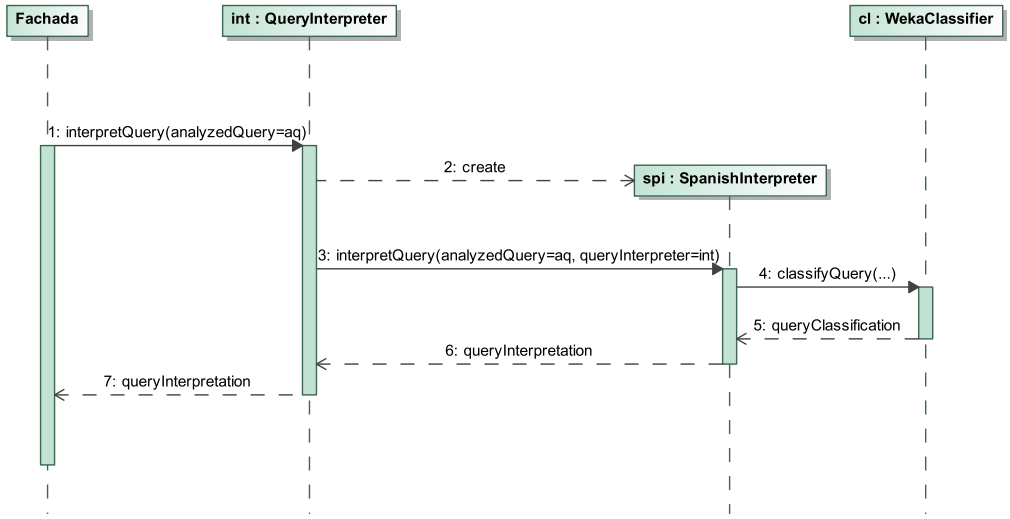


Figura 5.15: Diagrama de secuencia del intérprete.

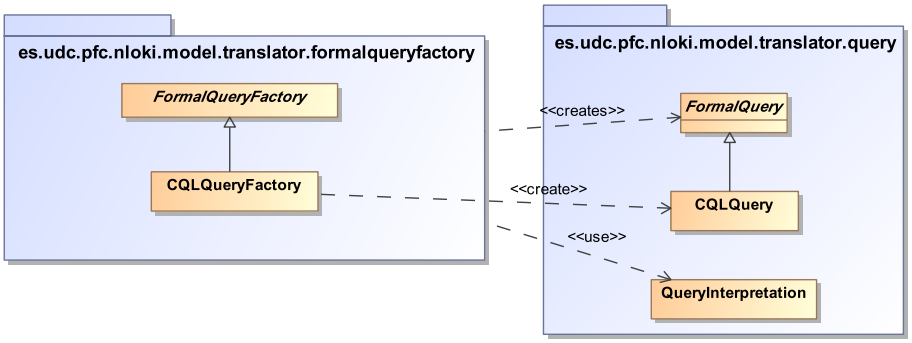


Figura 5.16: Diagrama de clases de la factoría de consultas en lenguaje formal.

para permitir cambios de estrategia de forma dinámica, de modo que una instancia del traductor no se encuentre restringida a tratar preguntas en un solo idioma.

Los diagramas de clases y secuencia para este componente pueden obervarse en las Figuras 5.13 y 5.15, respectivamente. Además, dado que la clasificación de la consulta se realiza también durante esta etapa, se presenta en este mismo apartado su diagrama de clases: Figura 5.14.

5.2.7. Factoría de consultas en lenguaje formal

El sistema desarrollado en este proyecto ha de poder soportar diversos lenguajes formales de salida, y no sólo CQL, por lo que desde el propio diseño debe facilitarse la extensión

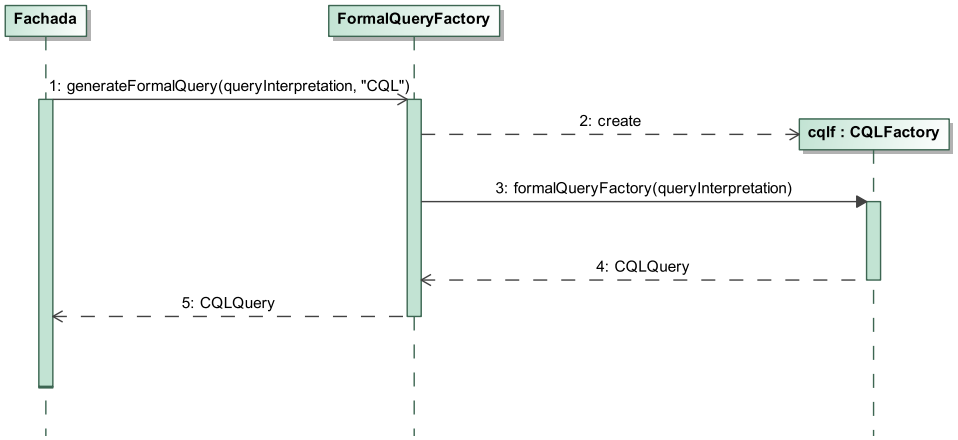


Figura 5.17: Diagrama de secuencia de la factoría de consultas en lenguaje formal.

de este componente. La opción más correcta para conseguir esto es mediante la aplicación del patrón *factoría*, tal y como puede observarse en las Figuras 5.16 y 5.17.

5.2.8. Traductor

Se trata de la *fachada* del sistema, mediante la cual se accede a toda la funcionalidad provista por los componentes comentados en apartados anteriores, abstrayendo la complejidad del núcleo.

En la Figura 5.18 puede verse el diagrama de clases para este componente. Por otro lado, en las Figuras 5.19 y 5.20 pueden observarse dos diagramas de secuencia: el primero de ellos muestra una ejecución del sistema en modo no interactivo (`mode=0`), mientras que el segundo muestra el caso complementario (`mode=1`).

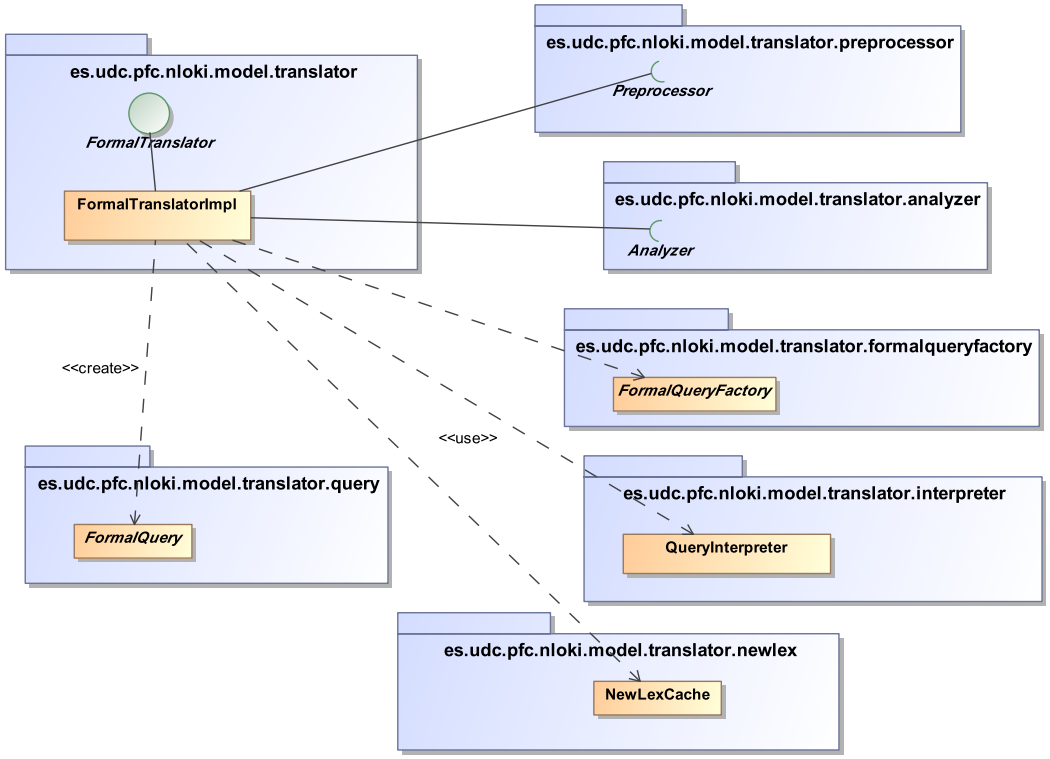


Figura 5.18: Diagrama de clases del traductor (fachada).

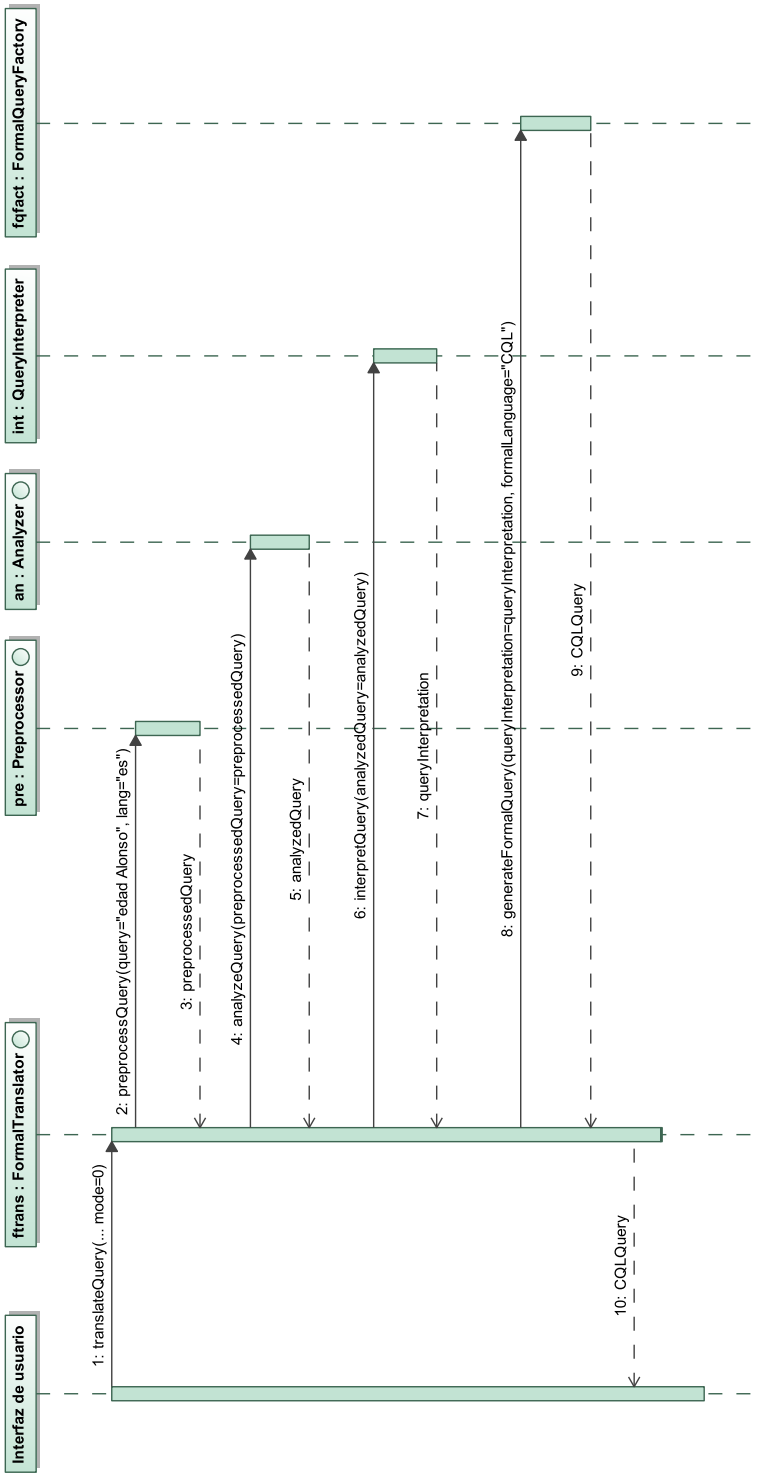


Figura 5.19: Diagrama de secuencia del traductor (fachada) en modo no interactivo.

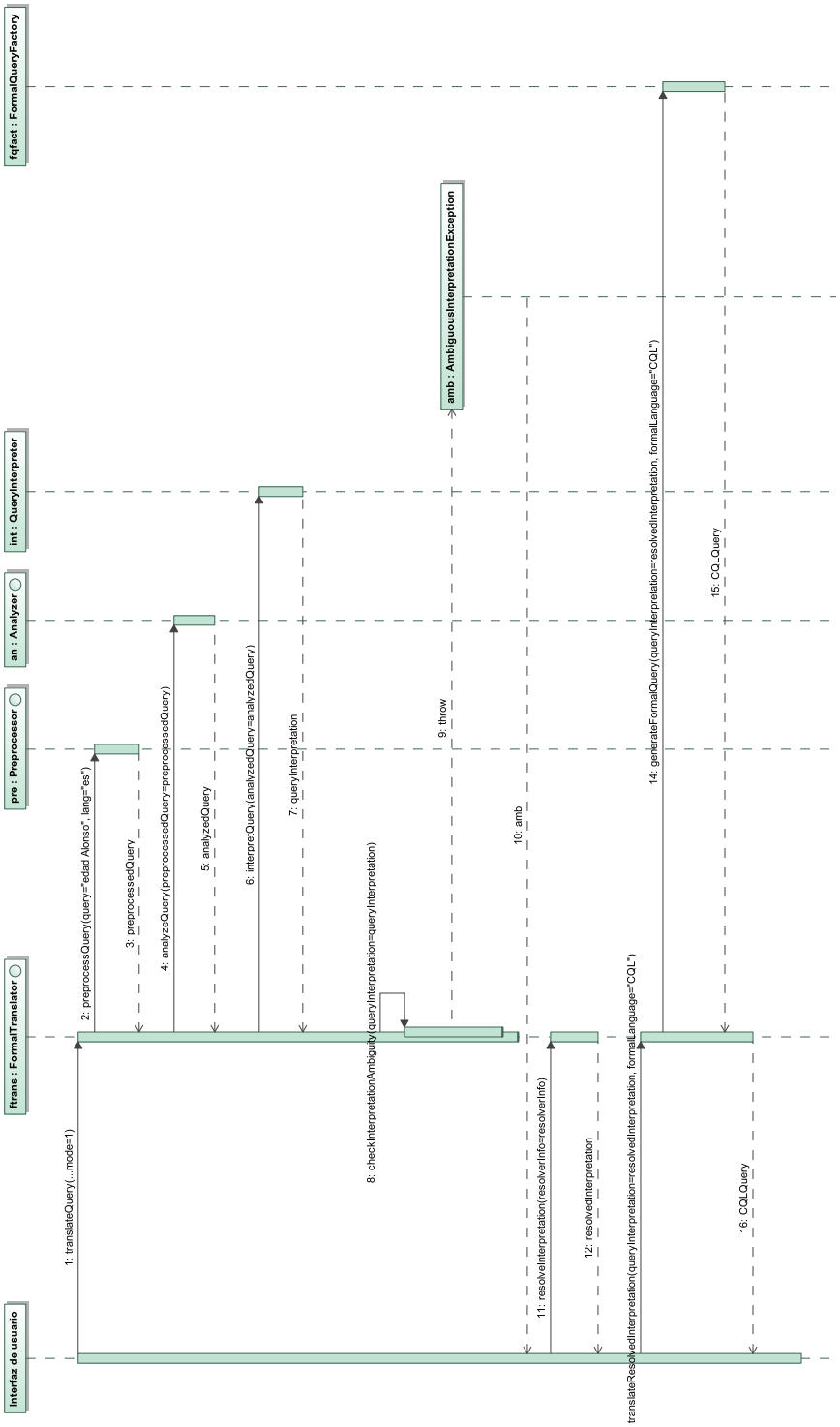


Figura 5.20: Diagrama de secuencia del traductor (fachada) en modo interactivo.

5.3. Diseño de la capa web

En este punto nos hemos ceñido al modelo propuesto por la implementación del patrón MVC de Struts 2. En la Figura 5.21 se puede observar el diseño de la capa web según las directrices establecidas por dicho modelo. Debemos precisar que, por cuestiones de claridad, no se han trazado líneas que indiquen la navegabilidad entre páginas puesto que prácticamente se puede ir de cualquier página a cualquier otra, dependiendo de la acción realizada, con una única excepción, que desde `error` sólo se puede navegar a `index`. De haber dibujado estas líneas, el diagrama hubiese resultado demasiado caótico.

En líneas generales, el estilo visual y manejo de las páginas que conforman esta capa se tratará de mantener lo más simplificado posible, siguiendo así la corriente minimalista contemporánea. Cabe destacar aquí, además, que en la página de resultados no sólo se proveerá al usuario con la información devuelta por CKB tras la ejecución de la consulta, si no que se añadirán otras dos secciones: en la primera se mostrará un fragmento del razonamiento seguido por el sistema al realizar la traducción de la consulta original al lenguaje formal; en la otra, se propondrá al usuario una pequeña lista de consultas relacionadas con la actual que pudieran ser de interés. De esta manera se provee al usuario de *feedback*, permitiéndole adquirir mayor conocimiento sobre el funcionamiento del sistema, lo cual le puede ser útil para formular de una forma más efectiva sus consultas.

En lo que respecta a las clases de esta capa, éstas se encuentran almacenadas en el paquete `es.udc.pfc.nloki.web.actions`.

5.4. Implementación

La implementación del sistema en su conjunto se llevó a cabo siguiendo las convenciones de programación para el lenguaje Java.[2] Por otro lado, si bien la correcta especificación de los resultados del análisis y diseño del sistema ya constituye de por sí buena parte de la documentación de éste, se ha documentado además el código fuente mediante Javadoc. Con estas medidas lo que se pretende es conseguir un código de calidad y mantenible, tanto por personas ajenas a su desarrollo que puedan hacer uso de él en un futuro (personal de Classora), como para el propio desarrollador del sistema.

En los apartados siguientes se detalla el proceso de implementación de los componentes principales del sistema, para los cuales dicha descripción resulta especialmente interesante.

[2] http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html

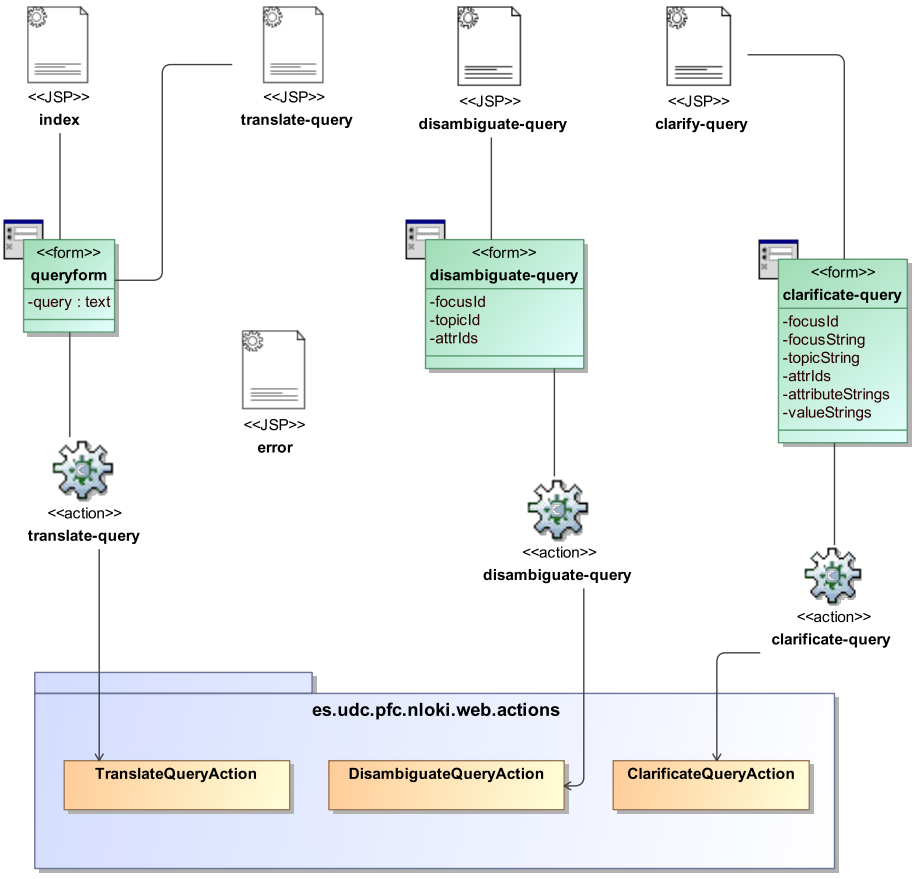


Figura 5.21: Estructura web.

5.4.1. Preprocesador

A pesar de que este componente esté actualmente implementado por un proceso base que no realiza ninguna acción sobre una consulta, y un proceso de corrección ortográfica que se apoya en el correspondiente servicio de Google, en un principio se ensayó con un tercer proceso: un *detector de idioma*, implementado a través de la librería `langdetect`.[3]

El objetivo de este componente era identificar automáticamente el idioma de la consulta lanzada originalmente por el usuario. Sin embargo, en nuestro contexto actual un sistema de este tipo no siempre es capaz de determinar el idioma correcto, como en el caso de la pregunta "edad Barack Obama", que no es detectada como español debido a lo exótico del nombre de la entidad a la que se refiere. Es por ello que se decidió desechar la idea y determinar el lenguaje de la consulta de entrada por otros medios. De este modo el idioma por defecto de la consulta será el preferido por el usuario según su navegador web y, de querer cambiar esto, se proveerá al usuario de un menú desplegable con los idiomas disponibles y en el cual podrá seleccionar el deseado. Cabe destacar que con esta aproximación se provee además al usuario de un mayor conocimiento sobre las capacidades del sistema al mostrarle los lenguajes soportados, lo que puede evitar frustraciones.

5.4.2. Analizador

En primer lugar, la implementación de este componente se apoya en las funciones provistas por las herramientas de PLN Treetagger y OpenNLP (véase Sección 4.2). Dado que éstas requieren que se las provea con modelos de datos del idioma específico con el que han de trabajar, estos modelos se almacenan en ficheros cuyo nombre sigue un patrón establecido y en el cual se indica el código del idioma para el que fueron construidos. De este modo, usando simplemente la información de entrada del sistema, en la cual se encuentra contenido este código, se puede variar dinámicamente en cada ejecución el idioma en el que operan estas herramientas. Además, esta misma técnica es empleada también para otros componentes como el corrector ortográfico del preprocesador, e incluso para las estrategias definidas en el intérprete.

En cuanto al propósito específico de cada herramienta, Treetagger se emplea para la obtención de *tokens*, *lemas* y *etiquetas morfosintácticas*, mientras que OpenNLP se utiliza para la detección de entidades con nombre. La utilidad de estos elementos se verá en los siguientes apartados, los cuales describen la implementación de los componentes que actúan a continuación del analizador.

[3] https://code.google.com/p/language-detection

Por otra parte, y tras resolver los problemas de integración de las herramientas de PLN descritos en el Apartado 4.2, se decidió emplear una aproximación basada en *pattern matching* para la detección de fechas, puesto que existe un conjunto finito y bien definido de maneras de referirse a una fecha de forma directa. Así, por ejemplo, podrían establecerse patrones como `día/mes/año` o `[número-de-día] de [nombre-de-mes] del año [número-de-año]`. Las fechas detectadas son sustituidas en la consulta por el término especial `<date>`, mientras que las fechas originales son almacenadas en una lista, desde donde podrán ser accedidas posteriormente.

Para aportar flexibilidad a esta aproximación, los patrones se almacenan en ficheros de texto agrupados por idioma, dado que aquellos escritos para una determinada lengua pueden no ser válidos para otra. De nuevo, estos ficheros contienen en su nombre el código del idioma para el que los patrones fueron construidos.

5.4.3. Clasificador

Dada la amplia variedad de algoritmos de clasificación disponibles en el *framework* de AA Weka, la elección de uno en particular se basó en el cumplimiento de dos premisas básicas: *rapidez de clasificación* y soporte para *entrenamiento incremental*. La necesidad de un proceso rápido de clasificación resulta evidente en un sistema interactivo como el nuestro: el usuario que desea resolver una duda lanzando su consulta al sistema no estará dispuesto a esperar demasiado tiempo para obtener su respuesta. Por otro lado, el hecho de que se permita la realización de un entrenamiento incremental facilita la tarea de mejora continua del sistema a través de la información obtenida a partir de los logs de uso, puesto que de ella se podrán extraer nuevas instancias de entrenamiento para el clasificador, y puede no resultar conveniente tener que *reentrenarlo* de nuevo con todas las instancias del conjunto de entrenamiento total. En base a esto, el algoritmo de clasificación escogido para nuestro sistema fue `NaiveBayes` [Manning and Schütze, 1999, Ch. 7], el cual cumple los requisitos establecidos inicialmente, si bien podría ser sustituido por cualquier otro que también satisficiera dichos requisitos.

La construcción del conjunto de entrenamiento se realizó de manera incremental en un proceso que consta de los siguientes pasos:

1. Se incluye una nueva *feature* (parámetro de clasificación) en el *diseño* del conjunto de entrenamiento.

2. Se construyen una serie de instancias de entrenamiento en base a las *features* consideradas, dando lugar a la *implementación* del conjunto de entrenamiento.

3. Se entrena el clasificador con el conjunto de entrenamiento construido en el paso anterior.

4. Se evalúa el clasificador contra las instancias del propio conjunto de entrenamiento: si el resultado es positivo, se detiene el proceso y se da por válido el diseño actual del conjunto; si el resultado es negativo, se introduce una nueva *feature* en el diseño, siempre y cuando los resultados obtenidos en el incremento actual sean mejores que los obtenidos en el anterior, puesto que en caso contrario se intercambiará la última *feature* introducida por la nueva, y se vuelve al paso 2.

La bifurcación en la lógica del proceso llevada a cabo durante el paso 4 pretende evitar el *overfitting* del clasificador, estado en el cual se vuelve demasiado restrictivo a la hora de clasificar las intancias en una clase u otra.

En nuestro caso concreto, la misión del clasificador es la de identificar el tipo de pregunta lanzada contra el sistema: Tipo 1 o Tipo 2 (véase Sección 1.3). En el proceso de construcción del conjunto de entrenamiento se comenzó utilizando como única *feature* el *string* de la propia consulta formulada por el usuario. Tras comprobar que este diseño era insuficiente, se fueron introduciendo incrementalmente el resto de *features*: etiquetas morfosintácticas de los términos de la consulta, número de palabras de la misma, número de atributos que aparecen en ella y, por último, el número de entidades que aparecen en la consulta. Tras la adición de esta última *feature*, el sistema alcanzó unos resultados satisfactorios durante la etapa de evaluación, por lo que se detuvo aquí el diseño del conjunto de entrenamiento. Cabe destacar además que durante parte de este proceso se sufrió de los efectos de un conjunto de entrenamiento *desbalanceado*, esto es, existían un buen número de instancias de Tipo 1 pero muy pocas de Tipo 2. Entrenado de este modo, el clasificador tenía la tendencia de clasificar las instancias de evaluación como de Tipo 1 la gran mayoría de las veces. La solución pasó por, lógicamente, *equilibrar* el conjunto de entrenamiento añadiendo más instancias de Tipo 2. Esto pudo realizarse gracias a la generación de un corpus adicional de preguntas desarrollado por personal externo al proyecto, para así evitar posibles contaminaciones, al que se le dieron las indicaciones pertinentes sobre la naturaleza de las preguntas que podrían realizar pero se dejó libertad a la hora de redactarlas. Dicho corpus es descrito más en detalle en la Sección 7.2.

Resulta interesante mencionar en este punto que en un principio el sistema disponía de dos clasificadores: el actual, y otro que tenía como objetivo determinar el *tipo de respuesta* de la pregunta formulada por el usuario. Dado el esfuerzo invertido en conseguir que el primer clasificador tuviese un buen rendimiento, y tras un tiempo de experimentación con el segundo, se decidió descartar éste último en favor de un *método basado en heurísticas*

para determinar el tipo de respuesta, a priori más sencillo que el proceso de entrenamiento del clasificador.

5.4.4. Intérprete

Dentro de este componente es posible implementar diferentes estrategias, cada una de ellas focalizada en el proceso de interpretación de consultas en un idioma concreto. Para conseguir que éstas sean intercambiables dinámicamente, aparte de usar el propio *patrón estrategia*, se utiliza un método de selección de la estrategia a emplear muy parecido al usado en el analizador para escoger los modelos de datos: los nombres de las estrategias deben ser de la forma `[código-de-idioma]Interpreter`, de modo que en `QueryInterpreter` se puede conocer el nombre exacto de la clase de estrategia a instanciar en base al idioma de la consulta de entrada.

La funcionalidad más compleja implementada en este componente es, sin duda, la extracción de predicados, que se realiza leyendo uno a uno, de izquierda a derecha, los lemas de los términos que constituyen la consulta. En un principio, rellenar estas estructuras con los datos obtenidos a través de los términos de la consulta y los conceptos del léxico puede suponer ya el primer escollo a salvar. Sin embargo, el verdadero problema en este punto tiene otro nombre: la *separación de predicados*.

Una manera rápida de resolver este problema es mediante la definición del tipo de palabra *delimitador*. Esta aproximación se basa en el uso de las etiquetas morfosintácticas obtenidas durante la etapa de análisis, y mediante las cuales se detecta si el lema actual es o no un delimitador de predicados, determinando así si es necesario dar por completado el predicado actual y añadirlo a la lista de predicados extraídos o, por el contrario, se continúa con la construcción del mismo. Cabe notar que esta decisión no se basa únicamente en el hecho de encontrar un delimitador, sino que depende también de otros factores relacionados con el predicado actual en construcción, como por ejemplo si en sus campos de `atributo` y `valor` ya existen elementos o si existe un contexto verbal (que describimos más abajo).

A causa de las propiedades del lenguaje natural descritas en la Sección 1.4, podríamos tener un número muy grande de maneras de construir un mismo predicado usando para ello expresiones sin demasiado en común. Esto, unido a que en diferentes contextos una misma expresión podría dar lugar a predicados totalmente diferentes, constituye un buen indicativo de la gran complejidad de la tarea que tenemos entre manos. El establecimiento de diferentes *contextos de interpretación* puede ayudarnos a mitigar algunos de estos problemas:

- *Contexto de la pregunta.* Se obtiene al considerar la unidad de conocimiento principal sobre la que se quieren obtener datos, lo cual permite centrar en mayor medida la búsqueda de los conceptos relevantes en el léxico para los términos de la consulta.
- *Contexto verbal.* Tener en cuenta los conceptos asociados a un verbo, como por ejemplo `fecha de nacimiento` y `lugar de nacimiento` para "nacer" en el contexto de la unidad de conocimiento `persona`, puede resultarnos de utilidad no sólo para el predicado que se esté construyendo en ese momento, sino para cualquiera de los subsiguientes, permitiendo así disponer de información útil que de otra manera no estaría presente según se fueran formando predicados.

Además, el uso de los lemas de las palabras cuando se buscan correspondencias en el léxico de los términos de la pregunta permite abordar la problemática derivada de la variabilidad lingüística que se puede dar en éstos, reduciendo en gran medida el número de conceptos a almacenar en el léxico y permitiendo, en definitiva, un comportamiento más robusto de este componente.

Si bien la lógica del intérprete es lo suficientemente compleja como para que no resulte interesante describirla aquí en su totalidad, en el Capítulo 6 se exponen con bastante detalle dos casos de uso del sistema en los cuales se hace hincapié en el funcionamiento de este componente.

5.4.5. Factoría de consultas en lenguaje formal

Un método similar al utilizado en el intérprete es empleado también dentro de este componente. En este caso ha de decidirse de forma dinámica qué clase específica de factoría ha de implementar el método de generación de la consulta en lenguaje formal a partir de los datos obtenidos del intérprete. Para ello, se establece que los nombres de las factorías concretas deben seguir el patrón `[lenguaje-formal]QueryFactory`, resultando así muy sencillo instanciar la factoría necesaria en cada caso conociendo únicamente el lenguaje de salida deseado.

5.4.6. Caché de nuevo léxico

Este componente, aunque no tan complejo como otros, cumple una función muy importante dentro del sistema: implementar la capacidad de aprendizaje. Para ello, en la clase `NewLexCache` (implementación de la caché) se almacenan instancias de `NewLex` (representación de un posible *nuevo concepto*), lo que nos permite almacenar los datos obtenidos a través de los diálogos de clarificación con el usuario, diálogos en los que normalmente

se hace referencia a conceptos que no existen todavía en el léxico del sistema. Dentro de esta caché se encuentra una lista en la que figuran todos los posibles nuevos conceptos, relacionados con elementos tales como atributos, unidades de conocimiento u otros conceptos, todos ellos datos obtenidos del contexto en el que apareció el término desconocido, asignándoseles además una *puntuación*. Esta puntuación, que comienza en 0, va aumentando de uno en uno cada vez que un usuario nombra el posible nuevo concepto en un diálogo de clarificación. Cuando esta puntuación sobrepasa un valor umbral definido en la propia caché de nuevo léxico, se procede a introducir este nuevo concepto en el léxico.

5.4.7. Capa vista

Durante la implementación de esta capa, se trató de dotar a las páginas de un diseño adaptativo para los diferentes formatos de pantalla que se pueden encontrar actualmente en los dispositivos con acceso a Internet, desde ordenadores a dispositivos móviles tales como *smartphones* o *tablets*, además de asegurar el soporte para los navegadores web más extendidos. Para ello se hizo uso del *front-end framework* Bootstrap[4] combinado con el tema visual Flat UI, [5] los cuales aportan todos los elementos necesarios para alcanzar estos objetivos, además de facilitar la construcción de una interfaz visualmente atractiva a la vez que sencilla. Por otra parte, se ofrece también soporte para *internacionalización*.

[4] http://twitter.github.io/bootstrap
[5] http://designmodo.github.io/Flat-UI

Capítulo 6

Ejemplos de funcionamiento

Hasta ahora hemos visto de qué modo fue construido el sistema, lo cual puede darnos bastantes pistas sobre cómo funcionaría al enviarle una serie de consultas una vez ya desplegado en un servidor web. Sin embargo, en este capítulo se quiere hacer hincapié en esto último: el propio funcionamiento del sistema. Así, para ilustrar el modo en que se procesan las consultas formuladas por el usuario, se exponen a continuación dos ejemplos concretos con consultas de Tipo 1 y Tipo 2 (asumiendo para ello que el sistema funciona en modo interactivo), lo que permitirá una mayor comprensión de los procesos involucrados y la complejidad de los mismos.

Cabe destacar en este punto que el sistema permite preguntas tanto *tipo enunciado*, como son las que se verán en los ejemplos descritos a continuación, como también en forma de *palabras clave* (p.ej. `edad Fernando Vázquez`), o incluso comenzar las preguntas con "entradillas" del tipo "Dime...", "Necesito saber...", etc., ofreciendo un comportamiento robusto en este sentido.

6.1. Ejemplo de Tipo 1: "¿Qué ocupación tiene Fernando?"

En la captura de pantalla mostrada en la Figura 6.1 se puede ver la apariencia de la página web de bienvenida a la aplicación, con la consulta de ejemplo ya introducida en la caja de texto colocada a tal efecto. A su lado, el desplegable en el que actualmente se lee "Español" permite seleccionar el idioma en el que la consulta está escrita, aunque por defecto se toma el del navegador (véase Sección 5.4.1), de forma que pueda ser tratada correctamente por el sistema.

Una vez que presionamos el botón `Enviar`, la consulta es introducida en el sistema traductor a través de su fachada (`FormalTranslator`) desde donde es enviada, en primer

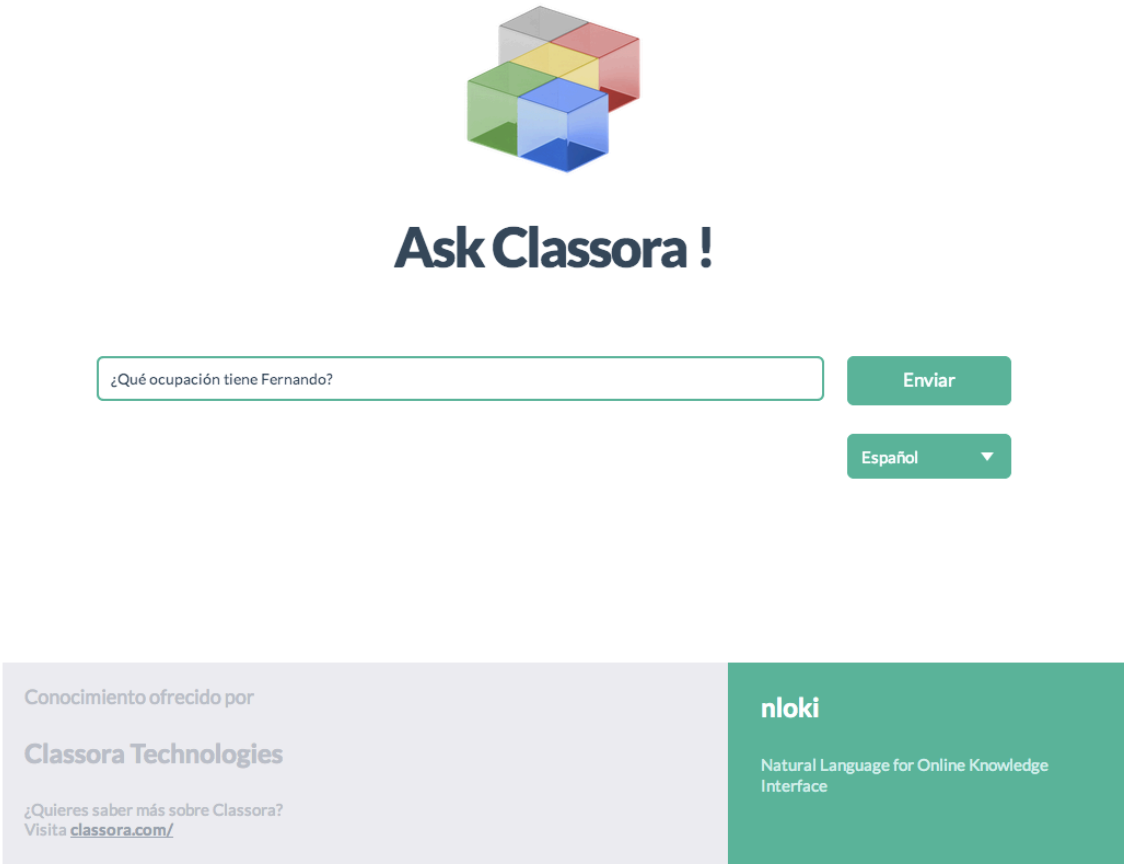


Figura 6.1: Página de inicio.

lugar, al preprocesador (`Preprocessor`). Allí se valida que carece de fallos ortográficos (a través de `GoogleSpellingChecker`), produciendo como salida una `PreprocessedQuery` en la que no se aprecian cambios entre la consulta original y la consulta procesada, y que entra directamente en el analizador (`Analyzer`). En esta etapa se realizan las siguientes acciones:

1. Se extraen los *tokens* {`¿`, `Qué`, `ocupación`, `tiene`, `Fernando`, `?`}.
2. Se obtienen sus etiquetas morfosintácticas: {`FS`, `INT`, `NC`, `VLfin`, `NP`, `FS`}.[1]
3. Se obtienen sus lemas: {`¿`, `qué`, `ocupación`, `tener`, `Fernando`, `?`}.
4. Se extraen las fechas mencionadas en la consulta (en este caso no hay) y se almacenan dentro de un objeto de tipo `QueryDates`.

Con todos estos datos se genera un objeto de tipo `AnalyzedQuery`, el cual sirve de entrada para la siguiente etapa a través de `QueryInterpreter`.

Esta nueva etapa, la interpretación de la consulta, comienza con la elección de la estrategia a emplear en base al idioma de entrada; en este caso `ESInterpreter`, puesto que la consulta está en español. Una vez que nos encontramos ejecutando la lógica de la estrategia, se siguen los siguientes pasos:

1. Se determina la unidad de conocimiento sobre la que trata la consulta o, dicho de otra manera, la unidad de conocimiento de la entidad sobre la que se quiere conocer algo (`focusedKUnit`). En este caso será `persona`, puesto que es el tipo de entidad al que hacen referencia todos los conceptos encontrados en la consulta.
2. Acto seguido se obtienen las entidades que aparecen en la consulta, sean del tipo que sean. En este caso se encuentran `Fernando Alonso` y `Fernando Vázquez`, conceptos obtenidos del léxico a partir del término `Fernando` presente en la consulta (cabe notar que en el léxico actual no existe ningún otro Fernando). Toda la información de este paso se almacena en un objeto `QueryEntities`.
3. Con la información contenida en los objetos de tipo `QueryDates` y `QueryEntities`, se sustituyen los lemas de la consulta de modo que en lugar de fechas y nombres de entidades aparezcan los elementos especiales `<date>` y `<entity>`, respectivamente. Se hace algo similar con las etiquetas morfosintácticas.
4. Generación de predicados (lista de `Predicate`). Se comprueban uno a uno los lemas de la consulta; en el caso de nuestro ejemplo:

[1] Símbolo de puntuación, pronombre interrogativo, nombre común, verbo, nombre propio y símbolo de puntuación, respectivamente.

- "¿". No se corresponde con ningún concepto del léxico ni, tras comprobar su etiqueta morfosintáctica (`FS`), constituye un tipo de palabra que sea necesario tratar.
- "`Qué`". Caso similar al anterior.
- "`ocupación`". Tras una búsqueda en el léxico se encuentra que, en el contexto de la unidad de conocimiento `persona`, este término está relacionado con el atributo `profesión`. Dicho atributo se introduce en el campo de posibles atributos del predicado que se está formando actualmente. El tipo de palabra no resulta relevante.
- "`tener`". Término no relacionado con ningún concepto, pero sí interesante desde el punto de vista de que se trata de un *verbo* (indicado por su etiqueta morfosintáctica). Los verbos cumplen la función de delimitadores de predicados, los cuales indican la finalización de la construcción del predicado actual siempre y cuando se cumplan unas condiciones. En este caso, la condición a cumplir sería simplemente que el predicado estuviese completo: que contuviese elementos tanto en su campo de *atributos* como en su campo de *valores*. Sin embargo, como éste no es el caso, se continúa con la formación del predicado actual.
- "`<entity>`". Este elemento especial indica que aquí había una entidad y, por lo tanto, debemos ir a buscar el nombre de entidad correspondiente en la lista de entidades contenida en el objeto de tipo `QueryEntities`. En este caso aparecen dos candidatos para esta posición, `Fernando Alonso` y `Fernando Vázquez`, por lo que ambos son introducidos en el campo `valor` del predicado actual.
- "`?`". Caso similar a "¿".

5. En base a los datos obtenidos hasta el momento, se clasifica la consulta mediante una llamada al clasificador (`Classifier`), el cual determina que se trata de una pregunta de Tipo 1, almacenando esta información en un objeto de tipo `Classification`.

6. Se obtienen todos los posibles *focos* de la pregunta en base al tipo de la misma. En este caso, por ser de Tipo 1, se recolectarán los atributos a los que hace referencia el único predicado que pudimos formar previamente, que en este caso es únicamente `profesión`, nuestro foco.

7. El *tema* de la pregunta habrá que buscarlo entre las entidades del tipo de la unidad de conocimiento especificada por `focusedKUnit`. En este caso tenemos dos posibilidades equiprobables: `Fernando Alonso` y `Fernando Vázquez`. Y así, con toda esta información se genera un objeto de tipo `QueryInterpretation`.

Figura 6.2: Diálogo de desambiguación.

Terminada la etapa de interpretación, y devolviendo el control de la ejecución a la fachada del sistema, se comprueba si existe algún dato ambiguo en el resultado de esta etapa. Como en este caso tenemos dos posibles temas para la consulta, se lanzará una excepción de tipo `AmbiguousInterpretationException`, de modo que sea interceptada por la capa web y se proceda a realizar el proceso de desambiguación a través del diálogo correspondiente, tal y como se puede apreciar en la Figura 6.2.

Una vez obtenida la respuesta del usuario, que en este caso supondremos que es `Fernando Alonso`, se vuelve a crear un objeto de tipo `QueryInterpretation`, en el que ahora figurará únicamente esta entidad como tema (lo cual es realizado de nuevo por el intérprete, invocado a través de la fachada).

Como último paso de traducción, y a través de la clase `FormalQueryFactory`, se crea una factoría específica para la generación de la consulta formal en el lenguaje CQL. Así, dado que la pregunta formulada por el usuario es de Tipo 1, el formato que se le dará será:

`atributo, entidad`

desde otro punto de vista, podría entenderse también como:

`foco, tema`

siendo el resultado obtenido tras la ejecución de esta etapa un objeto de tipo `CQLQuery` que contiene la salida deseada del traductor:

`profesión, Fernando Alonso`

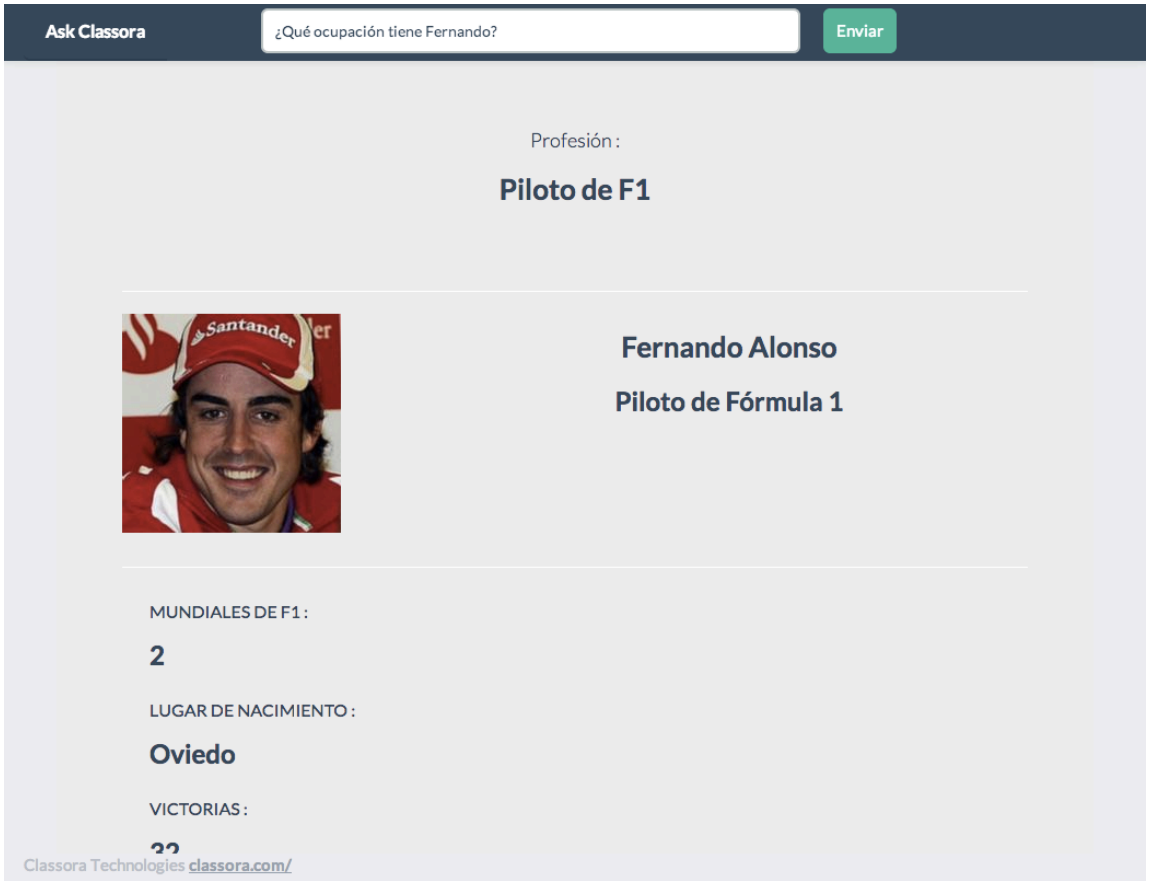


Figura 6.3: Página de resultados (parte superior).

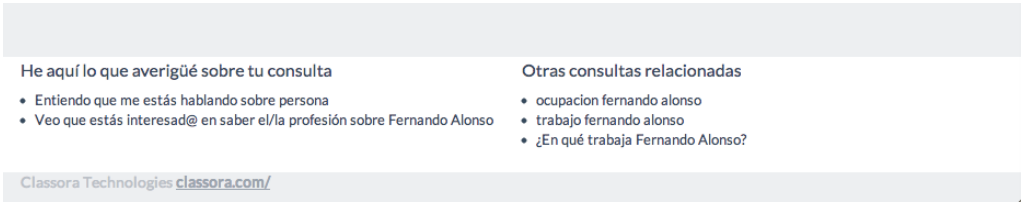


Figura 6.4: Página de resultados (parte inferior).

Finalmente, la consulta en lenguaje CQL es enviada a CKB a través de su API web, y tras parsear los datos formateados en XML devueltos por este servicio, se obtiene la página de resultados, tal y como se puede ver en las Figuras 6.3 y 6.4 (parte superior e inferior de dicha página, respectivamente).

6.2. Ejemplo de Tipo 2: “Mujeres nacidas en españa en 1990”

En este caso, el proceso es muy similar al descrito en la sección anterior, por lo que nos centraremos en aquellos aspectos en los que difieren ambos ejemplos.

De la etapa de preprocesamiento de la consulta, en la que se ejecuta el corrector ortográfico, se obtiene esta vez un objeto de tipo `PreprocessedQuery` en el que la consulta procesada difiere de la original en una única corrección: el término `españa` (incorrecto) es sustituido por `España` (correcto).

En la etapa de análisis nos encontramos esta vez con una fecha: el año `1990`. En este caso la expresión de la fecha coincidirá con el patrón `en [número-entero]`, sustituyendo ésta por el elemento especial `<date>` y almacenándola en la lista de fechas ubicada en el objeto de tipo `QueryDates`.

Llegados a la etapa de interpretación, y tras determinar el clasificador que la consulta de entrada es de Tipo 2, resulta especialmente interesante ver qué ocurre en este caso concreto durante la formación de predicados:

1. El primer lema, “`mujer`”, se corresponde con un concepto homónimo en el léxico, el cual está relacionado con el atributo de `persona género`. Y no sólo eso, sino que está relacionado también con otro concepto del léxico: `femenino`. De esta manera, con sólo mirar este término ya tenemos datos suficientes para formar el primer predicado, `genero = femenino`, que será introducido en la lista de predicados detectados. Debemos precisar en este punto que el operador de igualdad (`=`) es el empleado por defecto en todos los predicados, a no ser que algún término de la consulta especifique otro. Asimismo, si varios términos hacen referencia a distintos operadores para el mismo predicado, se obtendrá un solo operador fruto de la integración de todos ellos según unas reglas establecidas.

2. El término “`nacer`” no sólo se corresponde con varios conceptos del léxico en el contexto de `persona`: `lugar de nacimiento` y `fecha de nacimiento`, sino que es también un verbo (dato indicado por su etiqueta morfosintáctica). Así, dada dicha combinación de factores, no sólo podremos rellenar el campo `atributo` del predicado

actual con los conceptos asociados, sino que estos mismos conceptos pasan a formar parte de lo que llamamos *contexto verbal*. Este elemento nos permitirá “mantener en reserva” los dos atributos mencionados, de modo que puedan ser usados también en los posibles siguientes predicados a formar.

3. La preposición “`en`” no resulta relevante para el sistema en este punto.

4. A continuación no nos encontraremos directamente con la entidad “`España`”, si no con el elemento especial `<entity>`, por lo que para obtener el nombre de la entidad que ocupaba previamente esta posición debemos acceder a la lista de entidades del objeto `QueryEntities`. Dado que hasta ahora hemos estado hablando sobre *personas*, y lo que acabamos de encontrar es un *lugar*, se comprueba en la lista de atributos de `persona` si hay alguno cuyo tipo de valor asociado sea precisamente un lugar, obteniendo entonces `lugar de nacimiento`. Como este atributo ya aparecía en el campo correspondiente del predicado en construcción (por el término `nacer`), el sistema concluye que ha de ser éste el atributo buscado, y no `fecha de nacimiento`. De este modo el nombre de la entidad (“`España`”) pasará a formar parte del campo `valor` del predicado: `lugar de nacimiento = España`.

5. De nuevo nos encontramos con la preposición “`en`”, aunque en este caso sí resulta relevante, ya que actuará de delimitador de predicados. Así, puesto que el predicado que estábamos contruyendo hasta el momento parece estar ya completo, en este punto se insertará dicho predicado en la lista de predicados construidos, y se comenzará a “rellenar” un nuevo predicado, aunque partiendo del hecho de que ya disponemos de atributos candidatos especificados por el verbo anterior: `lugar de nacimiento` y `fecha de nacimiento`.

6. Por último, nos encontramos con el elemento especial `<date>`, así que iremos a buscar a la lista de fechas del objeto `QueryDates` aquella que se extrajo de esta posición de la consulta, la cual servirá como valor del predicado actual. En este mismo punto se puede determinar además el atributo concreto que ha de aparecer en este predicado mediante un método similar al explicado para la entidad “`España`”: dado que nos encontramos con una fecha, buscaremos en los atributos de la unidad de conocimiento `persona` si hay alguno cuyo tipo de valor asociado sea `fecha`. Tendremos así que `fecha de nacimiento` cumple esta condición, y puesto que ya se encontraba entre los candidatos a ser el atributo del predicado actual, se fija finalmente de este modo: `fecha de nacimiento = 1990`.

Cabe destacar, además, que en esta etapa las fechas son normalizadas a su equivalente en formato `dia/mes/año` siempre que esto sea posible. En este caso, por ejemplo, sólo

se menciona el año en la fecha encontrada, por lo que el formato de salida únicamente contendrá este dato pues ni el día ni el mes fueron especificados en la consulta.

Durante la etapa de generación de la consulta CQL, por tratarse en este caso de una Tipo 2, el formato de la misma será:

```
unidad de conocimiento*, predicado1, predicado2, ..., predicadoN
```

visto de otra manera:

```
foco*, predicado1, predicado2, ..., predicadoN)
```

El resultado es, por lo tanto, un objeto de tipo `CQLQuery` en el que se halla almacenada la consulta traducida:

```
persona*, genero = femenino, lugar de nacimiento = España,
fecha de nacimiento = 1990
```

En el caso de este ejemplo, desafortunadamente no podremos mostrar la captura de pantalla correspondiente a los resultados de la consulta pues, en el momento de redacción de la presente memoria, la API CQL correspondiente a las preguntas Tipo 2 se encuentra fuera de servicio desde hace varios días por razones externas a nosotros. Esperamos que en el momento de la defensa de este proyecto vuelva a estar en servicio.

Capítulo 7

Pruebas realizadas

En la mayoría de los casos, probar al cien por cien el correcto funcionamiento de un sistema software mínimamente complejo suele ser una tarea inviable. Sin embargo, es de vital importancia establecer un cierto número de pruebas de varios tipos diferentes para probar su funcionamiento, y el de sus componentes, en la mayoría de los casos en los que se pueda llegar a encontrar el sistema (que no en todos). Lo habitual suele ser realizar, por un lado, *pruebas unitarias y de integración* para probar los componentes que conforman el sistema, tanto individualmente como cuando interactúan entre sí; y por otro lado *pruebas de aceptación*, en las cuales el sistema en su conjunto se enfrentará a situaciones similares a las que se podría encontrar una vez desplegado.

7.1. Pruebas unitarias y de integración

Dado que es en la capa modelo donde reside la mayoría de la lógica del sistema, las pruebas se han focalizado sobre todo en esta parte. Para su automatización se ha hecho uso del framework JUnit, correspondiéndose los puntos de prueba con los servicios de esta capa. En este grupo se engloban tanto las clases que proveen del acceso a las entidades de la base de datos, como aquellas otras que representan las distintas etapas del núcleo del sistema. Así, para cada método de cada una de estas clases se define al menos un *test*, en el que se comprueba el correcto funcionamiento del mismo ante un caso de uso básico. Si el método en cuestión declara además el lanzamiento de una excepción, se crea un test para verificar que realmente se lanza ésta en el caso adecuado. Otros tests prueban el correcto funcionamiento de un método cuando se le pasan por parámetro valores "extremos", como por ejemplo `null` si el parámetro es un objeto, o -1 cuando es un número que hace referencia a un código de identificación (valor que suele considerarse un código inválido).

Para la capa web, se ha validado de forma manual tanto el correcto funcionamiento de cada componente (clase de acción) por separado, como su correcta integración con la capa modelo, todo ello a través de un servidor Jetty y haciendo uso de Java Platform Debugger Architecture (JPDA).[1]

7.2. Pruebas de aceptación

Estas pruebas fueron llevadas a cabo del mismo modo que las de integración de la capa web: de forma manual y con la aplicación corriendo desde un servidor Jetty. El conjunto de consultas de prueba fue extraído de un *corpus* de preguntas provisto por personal externo al proyecto para evitar su "contaminación", al que se le dieron las indicaciones adecuadas acerca de la naturaleza de las preguntas requeridas. El corpus resultante está formado por 202 instancias: 60 correspondientes a consultas de Tipo 1 abarcando los diferentes atributos disponibles sobre los que preguntar y 142 correspondientes a consultas de Tipo 2 cubriendo las diferentes combinaciones de atributos posibles en cuanto a las restricciones que se pueden imponer en la consulta, yendo desde 1 a 5 atributos.

Para la evaluación de los resultados obtenidos tras la ejecución de las consultas de prueba se utilizan dos métricas: el *porcentaje de traducciones correctas* y el *porcentaje de predicados correctos*. La primera de ellas, el porcentaje de traducciones correctas trata de cuantificar el número de resultados correctos desde la perspectiva del usuario, es decir, aquellos que dieron lugar a la respuesta buscada por éste. El porcentaje de predicados correctos, por su parte, hace referencia a las ocasiones en las que la extracción de predicados de la consulta se realizó de forma correcta, con independencia de la corrección de la forma final de la consulta en lenguaje formal. En este último caso, la correcta formación de los predicados pero la incorrecta construcción de la consulta de salida suele deberse a fallos del clasificador, puesto que el tipo de pregunta juega un papel esencial en dicha generación.

En la Figura 7.1 se muestran los resultados obtenidos en estas pruebas. Dada la diferencia en cuanto a la complejidad de las preguntas de uno u otro tipo, resultó interesante sacar no sólo mediciones que engloben al conjunto de consultas de entrada, sino, también, otras en las que se considere cada tipo por separado.

En vista de los resultados obtenidos pueden extraerse varias conclusiones:

- El porcentaje de aciertos global es lo suficientemente elevado como para validar el correcto desarrollo del sistema.

[1] http://docs.oracle.com/javase/6/docs/technotes/guides/jpda

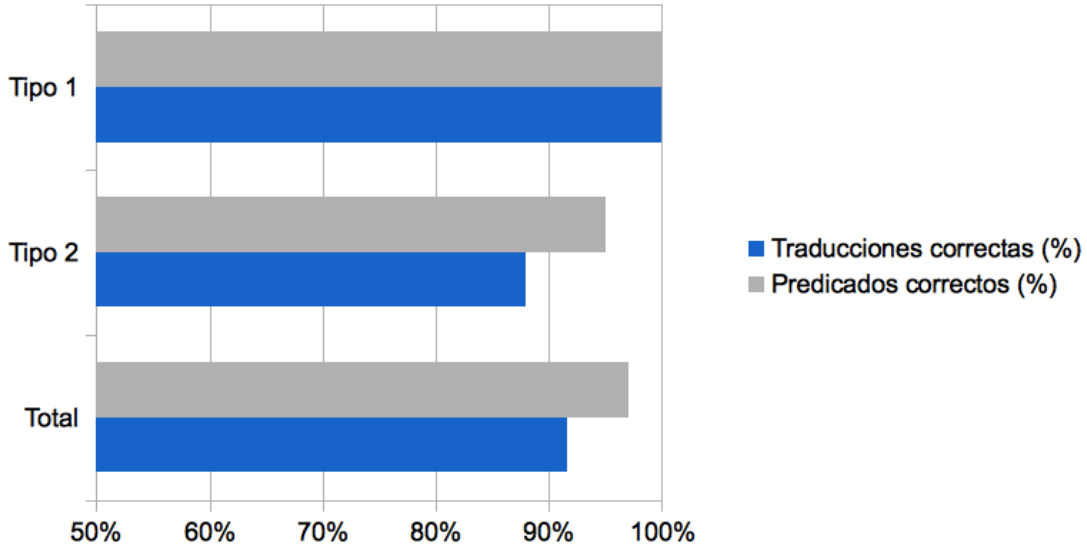


Figura 7.1: Resultados de las pruebas de aceptación.

- Se observa que el sistema resulta especialmente efectivo para las consultas de Tipo 1, más sencillas que las de Tipo 2, y con las que es menos probable sufrir los efectos de la expresividad del lenguaje natural.
- Se hace notar, en efecto, que existen casos en los que la clasificación fue errónea, por lo que el resultado final fue incorrecto a pesar de que la extracción de predicados se había realizado de manera satisfactoria. Estos casos podrían evitarse en un futuro mediante el uso de un conjunto de entrenamiento más amplio para el clasificador que, con los recursos disponibles actuales, no ha sido posible construir.

Por otra parte, un estudio en profundidad mediante JPDA de ciertos casos en los que el sistema falló en la formación de predicados, reveló que los componentes causantes de dichos fallos eran aquellos construidos mediante herramientas de PLN externas (véase Sección 4.2). Así, tanto los fallos en cuanto a la corrección ortográfica por parte del servicio de Google, como aquellos correspondientes al etiquetado morfosintáctico provisto por la librería Treetagger, fueron los causantes de que, en la mayor parte de estos casos, el intérprete del sistema fuera incapaz de formar correctamente los predicados. Como ejemplos, se presentan las siguientes dos consultas que hacen fallar a uno u otro componente:

- "`Necesito conocer las mujeres de Finlandia`". Inexplicablemente, el término "`Finlandia`" es corregido a "`Finlandesa`", el cual no se encuentra en el léxico, dan-

do así lugar a la consulta CQL:

```
persona*, genero = femenino
```

en la que no figura el atributo `lugar de nacimiento`, puesto que no se ha encontrado ningún gentilicio o nombre de país/región. Cabe destacar que este caso concreto podría subsanarse añadiendo al léxico el concepto "`finlandés`" y relacionándolo convenientemente con el concepto "`Finlandia`".

- "`¿Qué personas nacidas en España en 1986 son mujeres?`". En este caso, al término "`son`" le es incorrectamente asociada la etiqueta `NC` (Nombre Común) en lugar de `VSfin` (Verbo Ser), de tal forma que el intérprete no es capaz de separar los predicados `fecha de nacimiento = 1986` y `genero = femenino`, mezclándolos tal y como se puede observar en la consulta CQL de salida:

  ```
  persona*, lugar de nacimiento = España, genero = 1986
  ```

Cabe mencionar que durante el proceso de pruebas surgieron oportunidades para realizar pequeños ajustes en el intérprete, de modo que su tasa de aciertos pudiera ser mayor. Tras estas correcciones, era necesario volver a realizar todas las pruebas anteriores para confirmar que el ajuste no había afectado a los resultados obtenidos hasta el momento, lo que resultó en una etapa de pruebas de aceptación más larga de lo esperado.

Capítulo 8

Gestión del proyecto

En este capítulo se describe el plan de trabajo seguido durante el desarrollo del proyecto, ayudándonos para ello de un diagrama de Gantt, además de una breve gestión de los riesgos a los que está expuesto el proyecto.

8.1. Planificación

El plan de trabajo obedece a un modelo de desarrollo incremental, en el que se contempla la creación del sistema completo a través de un cierto número de iteraciones, en cada una de las cuales se obtiene una versión funcional preliminar de alguna parte del sistema. En nuestro caso, se ha dividido el proceso de desarrollo en dos iteraciones: desarrollo de la capa modelo y desarrollo de la capa web. Tras la finalización de la primera iteración, el sistema resulta completamente funcional, aunque carece de una interfaz de usuario mediante la que poder utilizarlo. Aquí es donde entra en juego la interfaz web, que sería desarrollada en la segunda iteración, y que además de permitir al usuario interactuar con el sistema, llevaría a cabo la comunicación con la CKB.

En la Figura 8.1 se puede observar el diagrama de Gantt para la planificación de este proyecto. Las tareas a las que se hace referencia en dicho diagrama son las siguientes:

- *Documentación previa.* Se trata de una fase previa al comienzo del desarrollo propiamente dicho. En ella se trata de preparar el terreno para las fases posteriores.
    - *Adquisición de bases teóricas.* Antes de abordar el desarrollo de cualquier sistema, es necesaria la adquisición de los conceptos previos más importantes del campo de aplicación. En esta etapa se estudia la bibliografía más relevante del campo para así manejar con soltura las bases teóricas del sistema a desarrollar.

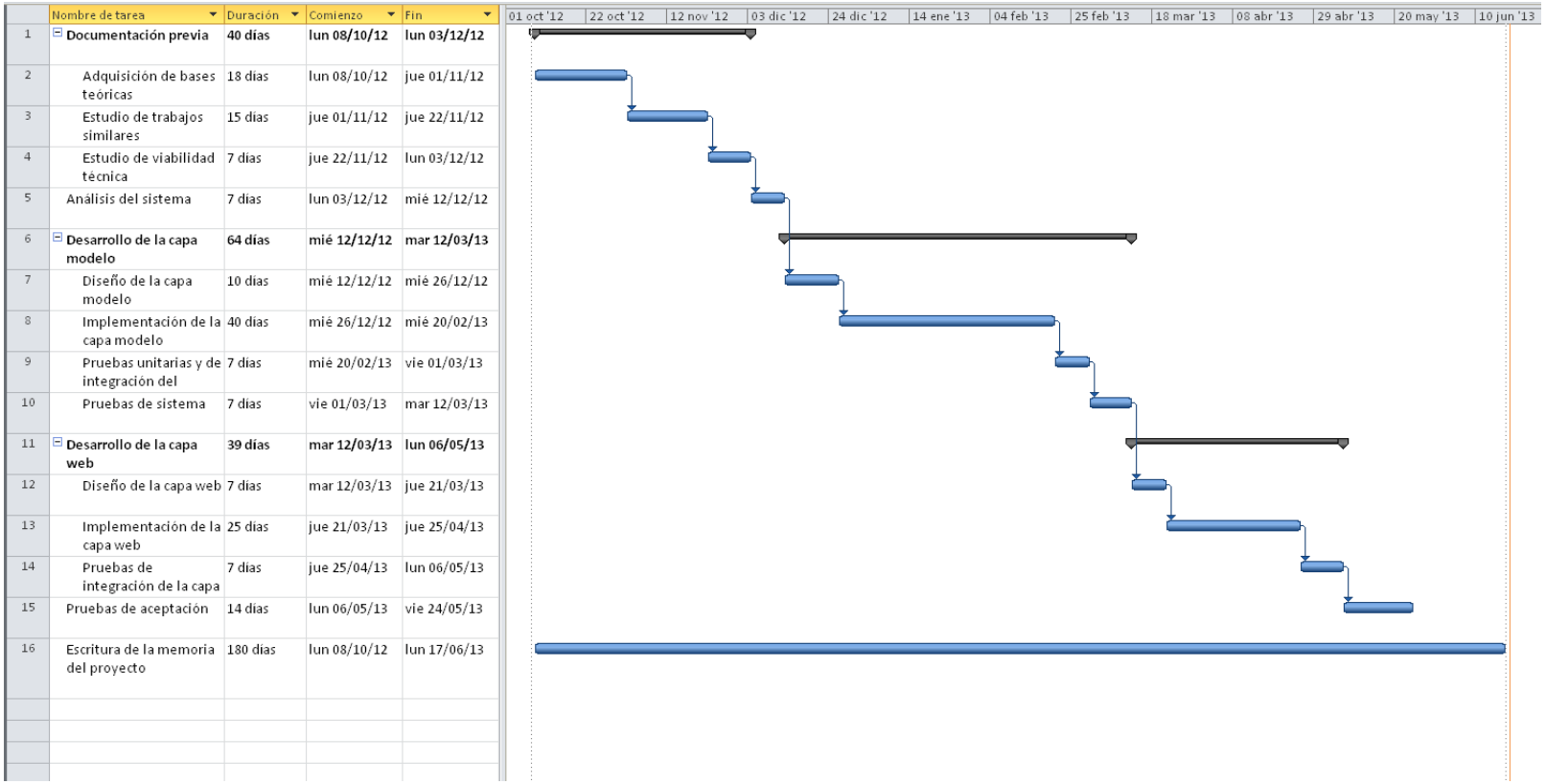
    - *Estudio de trabajos similares.* Una vez que se domina la teoría genérica a todos los sistemas de un determinado campo, se han de estudiar otros trabajos similares al que se va a realizar, las cuales podrían servir de modelo para el desarrollo de nuestro sistema.

- *Escritura de la memoria del proyecto.* Esta tarea se realiza a lo largo de todo el proyecto. En ella se va documentando cada una de las fases por las que va transcurriendo éste.

El resto de fases ya han sido descritas en apartados anteriores de esta memoria, por lo que simplemente las indicaremos:

- *Análisis del sistema.*
- *Desarrollo de la capa modelo.* Primera iteración.
    - *Diseño de la capa modelo.*
    - *Implementación de la capa modelo.*
    - *Pruebas unitarias y de integración del modelo.*
- *Desarrollo de la capa web.* Segunda iteración.
    - *Diseño de la capa web*
    - *Implementación de la capa web*
    - *Pruebas de integración de la capa web*
- *Pruebas de aceptación.*

En el diagrama de Gantt se puede observar cómo no existen paralelismos entre las tareas de desarrollo del sistema, aunque sí entre éstas y la escritura de la memoria. Esto es debido a que sólo se dispuso de un recurso de trabajo fijo: el proyectando. De disponerse de más trabajadores, y en caso de definirse de forma correcta todas las interfaces de los componentes que conforman el sistema, sería posible por ejemplo paralelizar las tareas de diseño e implementación de las capas modelo y web. Eso sí, de cualquier manera las pruebas de la capa web deberían posponerse hasta la finalización de las correspondientes a la capa modelo, puesto que se requiere que el núcleo del sistema funcione de forma correcta.



	Nombre de tarea	Duración	Comienzo	Fin
1	**Documentación previa**	**40 días**	**lun 08/10/12**	**lun 03/12/12**
2	Adquisición de bases teóricas	18 días	lun 08/10/12	jue 01/11/12
3	Estudio de trabajos similares	15 días	jue 01/11/12	jue 22/11/12
4	Estudio de viabilidad técnica	7 días	jue 22/11/12	lun 03/12/12
5	Análisis del sistema	7 días	lun 03/12/12	mié 12/12/12
6	**Desarrollo de la capa modelo**	**64 días**	**mié 12/12/12**	**mar 12/03/13**
7	Diseño de la capa modelo	10 días	mié 12/12/12	mié 26/12/12
8	Implementación de la capa modelo	40 días	mié 26/12/12	mié 20/02/13
9	Pruebas unitarias y de integración del	7 días	mié 20/02/13	vie 01/03/13
10	Pruebas de sistema	7 días	vie 01/03/13	mar 12/03/13
11	**Desarrollo de la capa web**	**39 días**	**mar 12/03/13**	**lun 06/05/13**
12	Diseño de la capa web	7 días	mar 12/03/13	jue 21/03/13
13	Implementación de la capa web	25 días	jue 21/03/13	jue 25/04/13
14	Pruebas de integración de la capa	7 días	jue 25/04/13	lun 06/05/13
15	Pruebas de aceptación	14 días	lun 06/05/13	vie 24/05/13
16	Escritura de la memoria del proyecto	180 días	lun 08/10/12	lun 17/06/13

Figura 8.1: Diagrama de Gantt.

8.2. Gestión de riesgos

Un *riesgo* es cualquier evento que, de suceder, tiene un impacto negativo sobre el proyecto, lo que normalmente conlleva retrasos indeseables y aumentos de coste con respecto a la planificación inicial. Dada la envergadura del proyecto que nos ocupa, tanto en recursos disponibles como en complejidad del mismo, se considerará suficiente con identificar, enumerar y analizar brevemente aquellos eventos que puedan constituir un riesgo. Los riesgos identificados son:

1. Indisposición del desarrollador por causas ajenas al proyecto.
2. Planificación poco realista.
3. Retraso en la provisión de recursos por terceras partes.
4. Cambio de herramientas a mitad de desarrollo.

Algunos de los riesgos enumerados son inevitables (1 y 3), mientras que el resto podrían ser mitigados:

- En el caso del riesgo número 2, una mayor experiencia en el ámbito de la planificación de proyectos podría mitigar el problema, a pesar de que es muy complicado obtener una planificación realista en cualquier caso.
- El último riesgo expuesto viene dado por la necesidad del uso de herramientas externas para la construcción de nuestro sistema. Si no se realizara un pequeño estudio previo de cada una de ellas, podría descubrirse muy tarde que una herramienta no realiza todas las funciones que se le requerían en un principio, o que no las realiza de acuerdo a nuestros estándares.

De cualquier manera, no existe un plan concreto de mitigación de riesgos, debiéndolos asumir en caso de que ocurran, pero prestando atención a los casos que se puedan evitar.

Parte IV

Resultados y conclusiones

Capítulo 9

Análisis de los resultados

En este capítulo se resumen las capacidades más destacadas del sistema desarrollado, se exponen algunas consideraciones a tener en cuenta de cara al despliegue del mismo en un entorno de producción, y se enumeran posibles mejoras futuras.

9.1. Prestaciones del sistema

Como funcionalidades más destacadas del sistema desarrollado en este proyecto cabe enumerar las siguientes:

- *Traducción de una consulta en lenguaje natural a lenguaje formal.* Ésta es la funcionalidad básica del núcleo del sistema.
- *Interfaz web* mediante la que interactuar con el sistema traductor de forma sencilla. En su diseño se ha tenido en cuenta, además, el *acceso mediante dispositivos móviles*, ajustándose la interfaz de forma automática al formato de la pantalla del dispositivo.
- Uso de los *diálogos de clarificación* para pedir ayuda al usuario en caso de no poder procesar una consulta de forma automática.
- *Capacidad de aprender* de las decisiones tomadas por el usuario en los diálogos de clarificación, de modo que se reduzca su frecuencia de aparición, siendo el sistema capaz de resolver futuras situaciones conflictivas de forma automática.
- *Dos modos de ejecución: automático y semiautomático.* En el primero el sistema tratará de resolver las ambigüedades que puedan surgir de forma automática, mientras que en el segundo buscará la ayuda del usuario mediante los diálogos de clarificación.

- *Feedback*. El usuario tendrá disponible, junto con los resultados de su consulta, una explicación del proceso de razonamiento seguido por el sistema para obtener dichos resultados. Además de esto, se mostrará una lista de consultas relacionadas con la actual y que por tanto puedan resultar de interés.

- *Flexibilidad del diseño*. El sistema facilita desde su propio diseño ampliaciones futuras en cuanto al número de lenguajes naturales de entrada y de lenguajes formales de salida.

9.2. Consideraciones para el despliegue final del sistema

La implementación actual del sistema, si bien pretende ser un prototipo, ofrece toda la funcionalidad especificada en los requisitos establecidos al inicio del proyecto. La condición de prototipo viene marcada por la cantidad de datos que alimentan a los distintos componentes del sistema en su estado actual, que en un entorno de producción será mucho mayor. Por una parte, el *conjunto de entrenamiento* actual del clasificador, aunque suficiente para nuestro prototipo, no lo es para condiciones normales de operatividad en el entorno de explotación, siendo necesaria la información que se irá almacenando en los logs, con el uso intensivo del sistema, para extender dicho conjunto. Por otra parte, al requerirse una construcción manual del *léxico*, es muy probable que su extensión inicial no sea la adecuada y se necesite un proceso incremental de mejora del mismo. A esto pueden ayudar en gran medida tanto los logs como las capacidades de aprendizaje del propio sistema, aunque, una vez más, tras un uso intensivo del mismo, el cual requiere tiempo.

9.3. Trabajo futuro

Para un sistema como el desarrollado existen muchas formas de mejora, puesto que se trata de un campo de aplicación muy extenso en el que se pueden dar infinidad de posibilidades de interacción entre el usuario y el sistema, causado en gran parte por el problema de la expresividad del lenguaje natural, tal y como se explicó en la Sección 1.4. A continuación se enumeran aquellas mejoras cuya implementación podría resultar especialmente interesante en un futuro cercano:

- *Soporte para preguntas encadenadas*. La implementación actual del sistema soporta únicamente *preguntas simples*, esto es, preguntas que no es posible dividir en otras más sencillas. Por su parte, las preguntas encadenadas son aquellas que es posible reformular como dos o más preguntas simples, encadenadas una detrás de la otra.

Un ejemplo de pregunta encadenada podría ser "¿Cuál es el número de habitantes de cada una de las provincias gallegas?", que podría dividirse en "¿Cuáles son las provincias gallegas?" seguido por (para cada provincia X) "¿Cuál es el número de habitantes de X?".

- *Soporte para preguntas con cláusulas relativas o pronombres relativos.* Dado que, por el momento, ni siquiera es posible expresar dichas estructuras lingüísticas en una consulta CQL, se decidió no complicar más la implementación inicial de nuestro sistema por el hecho de añadir esta funcionalidad. Un ejemplo de consulta de este tipo sería "Indica en un informe todos los estadios de Galicia, indicando su arquitecto, la ciudad de origen del arquitecto, y los habitantes de dicha ciudad".

- *Sistema de usuarios.* Podría ser interesante ofrecer a cada uno de los usuarios que utilicen este sistema un tratamiento personalizado, de forma que el sistema se amolde mejor a sus necesidades y preferencias particulares. Gracias a la implementación de esta mejora se podría ofrecer a los usuarios un conjunto de parámetros de configuración mediante los que modificar aspectos tales como el lenguaje de entrada por defecto, el modo de operación del sistema (semi/automático) o si en las preguntas de Tipo 1 ha de mostrarse un diálogo de clarificación si no se encuentra el foco en lugar de mostrar simplemente todos los atributos de la entidad encontrada.

- *Mejora continuada del sistema* a través de los logs, que permitirían extender tanto el léxico como el conjunto de entrenamiento del clasificador.

- *Disyuntiva de predicados.* Actualmente, los predicados obtenidos del análisis de las consultas se encadenan mediante operaciones lógicas AND (conjunciones), resultando en que las preguntas de Tipo 2 están formadas en parte por una conjunción de estas estructuras. Si bien es ésta la única manera soportada por el lenguaje CQL de encadenar predicados, podría pensarse en la posibilidad de añadir el encadenamiento mediante operaciones lógicas OR (disyunciones). De cualquier manera, se trataría de una mejora a largo plazo, puesto que no resultaría de utilidad hasta que se mejore de forma conveniente el propio lenguaje de consulta CQL, y esto es algo que depende de la empresa.

Capítulo 10

Conclusiones

En este último capítulo se pone en perspectiva el sistema desarrollado con respecto a los objetivos iniciales, además de dejar constancia de las lecciones aprendidas durante el proceso de desarrollo.

10.1. Contraste de objetivos

El objetivo principal de este proyecto, marcado al inicio de su desarrollo (ver Sección 1.3), era el de proveer a los usuarios de una ILN para el acceso al conocimiento contenido en CKB mediante una aplicación web. Siendo más específicos, el núcleo del problema a resolver era la traducción de una consulta en lenguaje natural a su equivalente en el lenguaje formal CQL. Pues bien, este objetivo se ha visto cumplido en el sistema desarrollado en este proyecto, en el que incluso se implementa un extra de flexibilidad al no restringirse las posibilidades de traducción a un solo lenguaje formal de consulta. Otros objetivos no considerados como tales anteriormente, a los que podríamos referirnos como secundarios, también se han visto cumplidos en mayor o menor medida.

En el enunciado del proyecto propuesto por Classora Technologies, se hablaba del tratamiento de cuatro unidades de conocimiento distintas: personas, empresas, países y ciudades. Esta cantidad podía redimensionarse de modo que constituyera un objetivo realista para el período de tiempo en el que se iba a desarrollar el proyecto. A este respecto se decidió trabajar únicamente con la unidad de conocimiento "persona" durante el proceso de desarrollo, aunque cabe destacar que nuestro sistema no está limitado por diseño a trabajar con este tipo de entidad, sino que está preparado para soportar cualquier otra siempre y cuando se complete la base de datos local de forma acorde.

Por otra parte, el segundo punto de la enumeración de posibles mejoras para el sistema (ver Sección 9.3), hace referencia a un aspecto del que ya se hablaba en tiempo futuro en el propio enunciado: el soporte para preguntas con cláusulas relativas. Por este motivo, no se consideró un objetivo primordial que cumplir con la implementación actual del sistema, pero sí resulta necesario tenerlo en cuenta para un posible desarrollo futuro.

Por último, el tiempo consumido por el desarrollo del proyecto fue ligeramente superior al estimado en el enunciado, en el que se preveía la finalización de este trabajo en un período de cuatro a seis meses. Si bien las estimaciones iniciales suelen ser ya algo optimistas, el hecho de que el comienzo del desarrollo de este proyecto ocurriera a tan sólo tres meses de la fecha de entrega más temprana permitió una mayor relajación del plan de desarrollo (ver Sección 8.1), al encontrarse la siguiente fecha de entrega mucho más espaciada en el tiempo, aunque por encima del límite establecido en la estimación inicial. De cualquier forma, cabe destacar que en todo momento se trató de cumplir con dicha estimación.

10.2. Lecciones aprendidas

Tres han sido las lecciones aprendidas más importantes durante el desarrollo de este proyecto:

1. *Estudio previo de las herramientas necesarias.* A pesar de tratarse de algo que ya se hizo durante el desarrollo del proyecto, en lo que se pretende incidir con este punto es en la extensión del estudio. No resulta suficiente ver sobre el papel las capacidades teóricas de una herramienta determinada. Es muy importante realizar pruebas de uso, podría incluso hablarse quizás de pruebas de *aceptación*, con dicha herramienta de modo que se pueda comprobar de primera mano si realmente cumple con las expectativas puestas sobre ella. En el caso concreto de este proyecto, no sólo se descubrieron tarde las dificultades de integración en una aplicación Java de la librería de PLN Freeling, sino que también resultó insuficiente la solución obtenida a través de OpenNLP para el etiquetador morfosintáctico (véase Sección 4.2).

2. *Desarrollo del proyecto en un entorno que tenga buen soporte.* Esta regla fue parcialmente seguida desde el inicio del proyecto. Desde un principio se trató de trabajar en un entorno similar al de Classora, para así aprovechar el soporte que se podría recibir por su parte. Sin embargo, el uso de Mac OS X como SO en el que desarrollar el sistema provocó contratiempos por el escaso soporte encontrado cuando se trató de integrar Freeling. Es por ello que se recurrió a la virtualización, tal y como se explicó en la Sección 4.4.

3. *Definir muy bien los objetivos perseguidos por el proyecto.* Tal y como se comentaba en la Sección 9.3, para un sistema como el desarrollado existen muchas formas de mejora. De hecho, de no adherirse lo suficiente el proceso de desarrollo a los objetivos iniciales del proyecto, es muy posible acabar malgastando esfuerzo en la implementación de funcionalidades y otros aspectos que no aporten demasiado al cumplimiento de los objetivos principales.

La importancia de estas lecciones viene dada por la cantidad de esfuerzo necesario para corregir los problemas que pueden surgir en caso de no seguirlas, y que por lo tanto conlleva un posible retraso respecto a la planificación inicial del proyecto.

Apéndice A

Índice de acrónimos

AA Aprendizaje Automático

ACE Attempto Controlled English

API Interfaz de Programación de Aplicaciones (*Application Programming Interface*)

BR *Búsqueda de Respuestas*

CKB Classora Knowledge Base

CQL Classora Query Language

DAO Data Access Object

FREyA Feedback, Refinement and Extended Vocabulary Aggregation

GPL GNU General Public License

IoC Inversión de Control (*Inversion of Control*)

IDE Entorno de Desarrollo Integrado (*Integrated Development Environment*)

ILN Interfaz de Lenguaje Natural

IR Recuperación de Información (*Information Retrieval*)

J2EE Java 2 Platform, Enterprise Edition

JPDA Java Platform Debugger Architecture

MVC Modelo Vista Controlador

ORM Mapeo Objeto Relacional (*Object–Relational Mapping*)

PLN Procesamiento del Lenguaje Natural

QuestIO Question-based Interface to Ontologies

REN Reconocimiento de Entidades con Nombre

SCV Sistema de Control de Versiones

SO Sistema Operativo

VM Máquina Virtual

WWW World Wide Web

Bibliografía

[Allen et al., 1996] Allen, J. F.; Miller, B. W.; Ringger, E. K.; and Sikorski, T. (1996). "A robust system for natural spoken dialogue". In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 62–70. Association for Computational Linguistics.

[Antoniou and Van Harmelen, 2004] Antoniou, G. and Van Harmelen, F. (2004). *Semantic Web Primer*. the MIT Press.

[ASF, 2013] The Apache Software Foundation (ASF). (2013). "ApacheLicense". http://www.apache.org/licenses.

[Biermann et al., 1983] Biermann, A. W.; Ballard, B. W.; and Sigmon, A. H. (1983). "An experimental study of natural language programming". *International journal of man-machine studies*, 18(1), pp. 71–87.

[Brooke, 1996] Brooke, J. (1996). "SUS-A quick and dirty usability scale". *Usability evaluation in industry*, 189, pp. 194.

[Cimiano et al., 2008] Cimiano, P.; Haase, P.; Heizmann, J.; Mantel, M.; and Studer, R. (2008). "Towards portable natural language interfaces to knowledge bases – The case of the ORAKEL system". *Data & Knowledge Engineering*, 65(2), pp. 325–354.

[CKB, 2013] Classora Knowledge Base (CKB). (2013). "Classora Knowledge Base". http://www.classora.com.

[Classora, 2012] Classora (2012). "Bases de conocimiento en Internet". http://blog.classora.com/2012/01/26/bases-de-conocimiento-en-internet/.

[Dale et al., 2000] Dale, R.; Moisl, H. L.; and Somers, H. L. (2000). *Handbook of natural language processing*. CRC Press.

[Damljanović, 2011] Damljanović, D. D. (2011). *Natural Language Interfaces to Conceptual Models*. PhD thesis, The University of Sheffield, Department of Computer Science.

[Fellbaum, 1998] Fellbaum, C. (1998). *WordNet - An Electronic Lexical Database*. MIT Press, Cambridge.

[Fernández et al., 2003] Fernández, S.; Graña, J.; and Sobrino, A. (2003). "Introducing FDSA (Fuzzy Dictionary of Synonyms and Antonyms): applications on information retrieval and stand-alone use". *Mathware & soft computing*, 10(2–3), pp. 57–70.

[Ferrández et al., 2009] Ferrández, O.; Izquierdo, R.; Ferrández, S.; and Vicedo, J. L. (2009). "Addressing ontology-based question answering with collections of user queries". *Information Processing & Management*, 45(2), pp. 175–188.

[Fuchs et al., 2006] Fuchs, N. E.; Kaljurand, K.; Kuhn, T.; Schneider, G.; Royer, L.; and Schröder, M. (April 2006). *Attempto Controlled English and the semantic web*. Deliverable I2D7, REWERSE Project, URL http://rewerse.net/deliverables/m24/i2-d7.pdf.

[Gamma et al., 1995] Gamma, E.; Helm, R.; Johnson, R.; and Vlissides, J. (1995). "Design patterns: Elements of reusable object-oriented software". *Reading: Addison-Wesley*, 49.

[Gao et al., 2011] Gao, M.; Liu, J.; Zhong, N.; Chen, F.; and Liu, C. (2011). "Semantic mapping from natural language questions to OWL queries". *Computational Intelligence*, 27(2), pp. 280–314.

[GNU Project, 2013] GNU Project. (2013). "GNU General Public License". http://www.gnu.org/licenses/gpl.html.

[Hirschman et al., 1999] Hirschman, L.; Light, M.; Breck, E.; and Burger, J. D. (1999). "Deep Read: A reading comprehension system". In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 325–332. Association for Computational Linguistics.

[Iskold, 2008] Iskold, A. (May 2008). *Semantic Search: The Myth and Reality*. ReadWriteWeb, URL http://www.readwriteweb.com/archives/semantic_search_the_myth_and_reality.php.

[Jurafsky and Martin, 2009] Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Pearson Prentice Hall, 2 edition.

[Kaufmann and Bernstein, 2007] Kaufmann, E. and Bernstein, A. (2007). "How useful are natural language interfaces to the semantic web for casual end-users?". In *The Semantic Web*, pages 281–294. Springer.

[Kaufmann et al., 2006] Kaufmann, E.; Bernstein, A.; and Zumstein, R. (2006). "Querix: A natural language interface to query ontologies based on clarification dialogs". In *5th International Semantic Web Conference (ISWC 2006)*, pages 980–981.

[Kupiec, 1993] Kupiec, J. (1993). "MURAX: A robust linguistic approach for question answering using an on-line encyclopedia". In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 181–190. ACM.

[Kwok et al., 2001] Kwok, C.; Etzioni, O.; and Weld, D. S. (2001). "Scaling question answering to the Web". *ACM Transactions on Information Systems (TOIS)*, 19(3), pp. 242–262.

[Linckels and Meinel, 2007] Linckels, S. and Meinel, C. (2007). *Semantic Interpretation of Natural Language User Input to Improve Search in Multimedia Knowledge Base.* it - Information Technologies, 49(1):40-48.

[Litman and Silliman, 2004] Litman, D. J. and Silliman, S. (2004). "ITSPOKE: An intelligent tutoring spoken dialogue system". In *Demonstration Papers at HLT-NAACL 2004*, pages 5–8. Association for Computational Linguistics.

[Lopez et al., 2012] Lopez, V.; Fernández, M.; Motta, E.; and Stieler, N. (2012). "PowerAqua: Supporting users in querying and exploring the Semantic Web". *Semantic Web*, 3(3), pp. 249–265.

[Lopez and Motta, 2004] Lopez, V. and Motta, E. (2004). "Ontology-driven question answering in Aqualog". In *Natural Language Processing and Information Systems*, pages 89–102. Springer.

[Manning and Schütze, 1999] Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing.* MIT press.

[Ogden and Bernick, 1997] Ogden, W. C. and Bernick, P. (1997). "Using natural language interfaces". *Handbook of human-computer interaction*, pages 137–161.

[Popescu et al., 2004] Popescu, A.-M.; Armanasu, A.; Etzioni, O.; Ko, D.; and Yates, A. (2004). "Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability". In *Proceedings of the 20th international conference on Computational Linguistics*, page 141. Association for Computational Linguistics.

[Prager, 2006] Prager, J. (2006). "Open-Domain Question–Answering". *Foundations and Trends in Information Retrieval*, 1(2), pp. 91–231.

[Simmons, 1965] Simmons, R. F. (1965). "Answering English questions by computer: a survey". *Communications of the ACM*, 8(1), pp. 53–70.

[Vicedo, 2003] Vicedo, J. (2003). "Recuperación de información de alta precisión: los sistemas de búsqueda de respuestas". *Sociedad Espanola para el Procesamiento del Lenguaje Natural (SEPLN), Alicante*.

[Vilares, 2005] Vilares, J. (2005). *Aplicaciones del procesamiento del lenguaje natural en la recuperación de información en español*. PhD thesis, Departamento de Computación, Universidade da Coruña, Spain.

[Vilares et al., 2008] Vilares, J.; Alonso, M. A.; and Vilares, M. (2008). "Extraction of complex index terms in non-English IR: A shallow parsing based approach". *Information Processing & Management*, 44(4), pp. 1517–1537.

[W3C, 2013] World Wide Web Consortium (W3C). (2013). "Guía Breve de Web Semántica". http://www.w3c.es/Divulgacion/GuiasBreves/WebSemantica.

[Wang et al., 2007] Wang, C.; Xiong, M.; Zhou, Q.; and Yu, Y. (2007). "Panto: A portable natural language interface to ontologies". In *The Semantic Web: Research and Applications*, pages 473–487. Springer.

[Webb and Webber, 2009] Webb, N. and Webber, B. (2009). "Special issue on interactive question answering: Introduction". *Natural Language Engineering*, 15(1), pp. 1.

[Winograd, 1972] Winograd, T. (1972). "Understanding natural language". *Cognitive Psychology*, 3(1), pp. 1–191.

[Woods et al., 1972] Woods, W.; Kaplan, R.; and Nash-Webber, B. (1972). "The lunar sciences natural language information system: final report. BBN Report 2378, Bolt, Beranek, and Newman". *Inc., Cambridge, Mass.*

[Zheng, 2002] Zheng, Z. (2002). "AnswerBus question answering system". In *Proceedings of the second international conference on Human Language Technology Research*, pages 399–404. Morgan Kaufmann Publishers Inc.