Mejora de la Comprensión Lectora mediante Analogías para la Inclusión

Trabajo de Fin de Grado
Curso 2018–2019

Autor
Irene Martín Berlanga
Pablo García Hernández

Director
Virginia Francisco Gilmartín
Gonzalo Rubén Mendez Pozo

Grado en Ingeniería de Software
Facultad de Informática
Universidad Complutense de Madrid

Mejora de la Comprensión Lectora mediante Analogías para la Inclusión

Trabajo de Fin de Grado en Ingeniería de Software
Departamento de Ingeniería de Software e Inteligencia Artificial

Autor
Irene Martín Berlanga
Pablo García Hernández

Director
Virginia Francisco Gilmartín
Gonzalo Rubén Mendez Pozo

Dirigida por el Doctor
Virginia Francisco Gilmartín
Gonzalo Rubén Mendez Pozo

Grado en Ingeniería de Software
Facultad de Informática
Universidad Complutense de Madrid

16 de marzo de 2019

Autorización de difusión

Los abajo firmantes, matriculados en el Grado de Ingeniería de Software de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo de Fin de Grado: "Mejora de la Comprensión Lectora mediante Analogías para la Inclusión", realizado durante el curso académico 2018-2019 bajo la dirección de Virginia Francisco Gilmartín y Gonzalo Rubén Mendez Pozo en el Departamento de Ingeniería de Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Nombre Del Alumno

16 de marzo de 2019

Dedicatoria

Texto de la dedicatoria...

Agradecimientos

Texto de los agradecimientos

Resumen

Resumen en español del trabajo

Palabras clave

Máximo 10 palabras clave separadas por comas

Abstract

Abstract in English.

Keywords

10 keywords max., separated by commas.

Índice

Índice de figuras

Índice de tablas

Índice de Listados

Chapter 1

Introduction

Introduction to the subject area.

Capítulo 1

Introducción

El español, hoy en día es la segunda lengua más hablada del mundo y actualmente más de 90000 palabras forman el castellano. Se trata de una lengua con multitud de palabras, y que dependiendo del contexto en el que se encuentre, puede tener múltiples significados. Si esto puede suponer una complicación, por ejemplo para una persona que no tiene ningún trastorno cognitivo, para ciertos colectivos de la sociedad, lo es aún mucho más afectándoles en su vida cotidiana, profesional o personal.

A esto se suma, que nuestra lengua española, con motivo de los nuevos avances en tecnología y nuevos hábitos que van surgiendo, han hecho que el lenguaje haya tenido que evolucionar adecuándose a la misma evolución de la sociedad. En la sección 1.1 se explicará con más detalle este problema que afecta a una gran parte de la sociedad, y que no disponen de ninguna herramienta para entender ciertos conceptos complejos. Nuestro objetivo es ofrecer un servicio accesible que defina palabras complejas de una manera clara, empleando para ello palabras más sencillas y que en la sección 1.2 se explicarán todos los objetivos tanto tecnológicos como académicos que los integrantes que desarrollan dicho trabajo se han propuesto. Por útlimo, en la sección 1.3 vendrá explicada la estructura de dicho documento.

1.1. Motivación

En nuestra sociedad, existen ciertos colectivos como pueden ser inmigrantes, personas con algún tipo de trastorno cognitivo, ancianos, analfabetos funcionales, niños, etc... que tienen dificultad para aprender conceptos complejos. Existen multitud de palabras cuyo significado es bastante complicado de explicar de una manera sencilla, por lo que una solución para que cualquier persona los pueda comprender es hacer uso de metáforas o analogías en las que intervengan palabras conocidas para los usuarios. Por ejemplo, para explicar

que es un selfi, se puede decir que un “selfi es como una fotografía". De esta forma, se puede asimilar el concepto de una manera más rápida y sencilla haciendo así que la dificultad para entender conceptos complejos no suponga una limitacion en la vida cotidiana, en la forma de relacionarse con otros individuos, en la vida profesional e incluso la vida personal. Por ejemplo, una persona que sea analfabeta emocional puede tener limitaciones al ver un programa de televisión, leer un manual técnico para realizar su trabajo, utilizar el teléfono móvil, etc...

Para ayudar principalmente a estas personas a que puedan entender el significado de cualquier palabra, y de esta forma superar algunas de sus limitaciones, se va a desarrollar una aplicación que permita definir palabras complejas mediante comparaciones con otras más fáciles ya conocidas por ellos. Por ejemplo, si se quiere explicar una palabra compleja como puede ser piraña, se puede describir utilizando conceptos más simples de la siguiente manera: *“Una piraña nada como un pez y es agresiva como un león”*. Mediante esta comparación, alguien que desconozca completamente el significado de *piraña*, puede hacerse una idea muy aproximada de lo que es.

1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es crear una aplicación web basada en servicios que dada una palabra compleja para el usuario devuelva una definición de dicha palabra mediante símiles, analogías o metáforas que empleen palabras más sencillas y conocidas para el usuario. Se utilizarán técnicas centradas en el usuario para diseñar la interfaz y así conseguir una aplicación usable y que se adapte a las necesidades y limitaciones de los potenciales usuarios finales. Los objetivos tecnológicos a alcanzar en este Trabajo de Fin de Grado son:

- La aplicación estará construida con servicios web que la doten de funcionalidad a la aplicación para que sean reutilizables en otras aplicaciones y se puedan adaptar a las distintas necesidades de los usuarios finales.
- Los servicios web desarrollados estarán disponibles en una API pública para que todo el mundo pueda utilizarlos.
- La aplicación se construirá de manera incremental, añadiéndole valor al producto poco a poco.

Por último, no se deben de olvidar los objetivos académicos de este trabajo:

- Poner en práctica los conocimientos adquiridos durante el Grado y ampliar nuestros conocimientos en distintas áreas.

Alcanzando los objetivos anteriormente descritos, se conseguirá obtener un producto de calidad, con una gran utilidad tanto social como académica, que puede ayudar a mucha gente a aprender ciertos conceptos de nuestro idioma de una manera más sencilla.

1.3. Estructura de la memoria

En el **capítulo dos** se presenta el Estado de la Cuestión, en el que se explicará que es la Lectura Fácil y como se aplica y se introducirán los conceptos de Procesamiento del Lenguaje Natural (PLN) y algunas herramientas que sirven para PLN, además se hablará de figuras retóricas y servicios web, en especial qué son, su arquitectura y las ventajas y desventajas de su uso.

En el **capítulo tres** se describe el trabajo realizado por cada uno de los autores.

Capítulo 2

Estado de la Cuestión

Para que un contenido ilustrativo o en formato de texto sea sencillo de entender existe una adaptación llamada lectura fácil cuyo objetivo es facilitar la accesibilidad al mismo. En la sección 2.1 de este capítulo se explicará qué es la lectura fácil y algunas pautas que se pueden seguir para escribir correctamente un texto en lectura fácil. Por otro lado, para que las definiciones de las palabras difíciles sean fácilmente comprensibles para el usuario, se utilizarán figuras retóricas con las que se compararán conceptos complejos con otros más sencillos. De esta manera el usuario se podrá hacer una idea de sus características principales. En la sección 2.2 se hablará de las figuras retóricas y se explicarán los tres tipos fundamentales que se van a utilizar para la realización de este trabajo. Pero para poder plasmar todo esto en el trabajo realizado y conseguir una funcionalidad, se deberán usar servicios web, los cuáles se han convertido en una tecnología cotidiana para la mayoría de los usuarios sin ser estos mismos conscientes de su uso. En la sección 2.3 se explicará detalladamente que es un servicio web, los tipos que existen, sus características principales, su arquitectura y las ventajas de ser utilizados así como sus desventajas. Por último, las comparaciones entre conceptos más complejos con otros más sencillos, se construirán a través de conceptos relacionados con la palabra que el usuario busque, es por ello que en la sección 2.4 se explicará en qué consiste el Procesamiento del Lenguaje Natural, los tipos de redes semánticas que existen y las diversas aplicaciones que actúan como redes semánticas y que son capaces de procesar el Lenguaje Natural.

Figura 2.1: Logo Lectura Fácil

2.1. Lectura Fácil

Se llama lectura fácil[1] a aquellos contenidos que han sido resumidos y reescritos con lenguaje sencillo y claro, de forma que puedan ser entendidos por personas con discapacidad cognitiva o discapacidad intelectual. Es decir, es la adaptación de textos, ilustraciones y maquetaciones que permite una mejor lectura y comprensión. Este trabajo se va a centrar en la lectura fácil aplicada a textos.

La lectura fácil surgió en Suecia en el año 1968, donde se editó el primer libro en la Agencia de Educación en el marco de un proyecto experimental. A continuación, en 1976, se creó en el Ministerio de Justicia un grupo de trabajo para conseguir textos legales más claros. En 1984 nació el primer periódico en lectura fácil, titulado "8 páginas", que tres años más tarde, en 1987, se publicó de forma permanente en papel hasta que empezó a editarse en la web. En el año 2013, en México se produce la primera sentencia judicial en lectura fácil[2]. En la actualidad, podemos distinguir los documentos en lectura fácil gracias al logo de la Figura 2.1.

Los documentos escritos en Lectura Fácil (**?**) son documentos de todo tipo que siguen las directrices internacionales de la IFLA[3] y de Inclusion Europe[4] en cuanto al contenido y la forma. Algunas pautas a seguir para escribir correctamente un texto en Lectura Fácil son (**?**):

- Evitar mayúsculas fuera de la norma, es decir, escribir en mayúsculas sólo cuando lo dicten las reglas ortográficas, como por ejemplo, después de un punto o la primera letra de los nombres propios.

- Deben evitarse el punto y seguido, el punto y coma y los puntos

[1] https://www.discapnet.es/areas-tematicas/diseno-para-todos/accesibilidad-de-comunicacion/lectura-facil

[2] https://dilofacil.wordpress.com/2013/12/04/el-origen-de-la-lectura-facil/

[3] International Federation of Library Associations and Institutions

[4] Una asociación de personas con discapacidad intelectual y sus familias en Europa

suspensivos. El punto y aparte hará la función del punto y seguido.

- Evitar corchetes y signos ortográficos poco habituales, como por ejemplo: %, & y /.

- Evitar frases superiores a 60 caracteres y utilizar oraciones simples. Por ejemplo, la oración *Caperucita ha ido a casa de su abuela y ha desayunado con ella* es mejor dividirla en dos oraciones simples: *Caperucita ha ido a casa de su abuela* y *Caperucita ha desayunado con ella.*

- Evitar tiempos verbales como: futuro, subjuntivo, condicional y formas compuestas.

- Utilizar palabras cortas y de sílabas poco complejas. Por ejemplo: casa, gato, comer o mano.

- Evitar abreviaturas, acrónimos y siglas.

- Alinear el texto a la izquierda.

- Incluir imágenes y pictogramas a la izquierda y su texto vinculado a la derecha.

- Evitar la saturación de texto e imágenes.

- Utilizar uno o dos tipos de letra como mucho.

- Tamaño de letra entre 12 y 16 puntos.

- Si el documento está paginado, incluir la paginación claramente y reforzar el mensaje de que la información continúa en la página siguiente.

Se debe también hacer hincapié en la distinción entre palabras fáciles y complejas (**?**), puesto que son de gran importancia para la lectura fácil. Las palabras complejas son aquellas que no se utilizan a menudo, como por ejemplo: melifluo o inefable. Es por ello que este tipo de palabras deben estar totalmente descartadas en la lectura fácil, y en su lugar debemos introducir palabras fáciles, que son aquellas que se utilizan asiduamente. La RAE (Real Academia Española) dispone de un documento con las mil palabras más usadas[5].

[5] http://corpus.rae.es/lfrecuencias.html

2.2. Figuras retóricas

Las figuras literarias (o retóricas) se podrían definir (**?**) como formas no convencionales de utilizar las palabras, de manera que, aunque se emplean con sus acepciones habituales, se acompañan de algunas particularidades fónicas, gramaticales o semánticas, que las alejan de ese uso habitual, por lo que terminan por resultar especialmente expresivas. Según la RAE (Real Academia Española)[6],*"la retórica es el arte de bien decir, de dar al lenguaje escrito o hablado eficacia bastante para deleitar, persuadir o conmover"*. La metáfora, el símil y la analogía se basan en la comparación de dos conceptos (**?**): el origen (o tenor), que es el término literal (al que la metáfora se refiere) y el de destino (o vehículo), que es el término figurado. La relación que hay entre el tenor y el vehículo se denomina fundamento. Por ejemplo, en la metáfora *Tus ojos son dos luceros*, *ojos* es el tenor, *luceros* es el vehículo y el fundamento es la belleza de los ojos.

En este trabajo se van a utilizar tres tipos de figuras retóricas (**?**):

- Metáfora: Utiliza el desplazamiento de características similares entre dos conceptos con fines estéticos o retóricos. Por ejemplo, cuando el tiempo de una persona es muy preciado se dice: "Mi tiempo es oro".
- Símil: Realiza una comparación entre dos términos usando conectores (por ejemplo, como, cual, que, o verbos). Por ejemplo, cuando nos referimos a una persona que es muy corpulenta, se dice: "Es como un oso", ya que los osos son muy grandes.
- Analogía: Es la comparación entre varios conceptos, indicando las características que permiten dicha relación. En la retórica, una analogía es una comparación textual que resalta alguna de las similitudes semánticas entre los conceptos protagonistas de dicha comparación. Por ejemplo: "Sus ojos son azules como el mar".

2.3. Servicios Web

Para definir el concepto de servicio web de la forma más simple posible, se podría decir que es una tecnología que utiliza un conjunto de protocolos para intercambiar datos entre aplicaciones, sin importar el lenguaje de programación en el cual estén programadas o ejecutadas en cualquier tipo de plataforma[7]. Según el W3C (*World Wide Web Consortium*)[8],*"un servicio web es un sistema software diseñado para soportar la interacción máquina-a-máquina, a través de una red, de forma interoperable"*.

[6] https://dle.rae.es/?id=WISC3uX
[7] http://www.jtech.ua.es/j2ee/publico/servc-web-2012-13/sesion01-apuntes.html
[8] https://www.w3.org/

Las principales características de un servicio web son (**?**):

- Es accesible a través de la Web. Para ello debe utilizar protocolos de transporte estándares como HTTP, y codificar los mensajes en un lenguaje estándar que pueda ser accesible por cualquier cliente que quiera utilizar el servicio.
- Contiene una descripción de sí mismo. De esta forma, una aplicación web podrá saber cual es la función de un determinado Servicio Web, y cuál es su interfaz, de manera que pueda ser utilizado de forma automática por cualquier aplicación, sin la intervención del usuario.
- Debe ser localizado. Debe tener algún mecanismo que permita encontrarle. De esta forma tendremos la posibilidad de que una aplicación localice el servicio que necesite de forma automática, sin tener que conocerlo previamente el usuario.

2.3.1. Tipos de Servicios Web

Los servicios web pueden definirse tanto a nivel conceptual como a nivel técnico. A nivel técnico se pueden diferenciar dos tipos de servicios web (**?**):

- Servicios web SOAP *(*Simple Object Access Protocol): SOAP es un protocolo basado en XML para el intercambio de información entre ordenadores. Normalmente utilizaremos SOAP para conectarnos a un servicio e invocar métodos remotos[9]. Los mensajes SOAP tienen el formato representado en el Listado 2.1, donde podemos ver un ejemplo para reservar un vuelo y está formado por los siguientes campos:
  - <Envelope>: elemento raíz de cada mensaje SOAP. Contiene dos elementos:
    - <Header>: es un elemento opcional que se utiliza para indicar información acerca de los mensajes SOAP. En el ejemplo del Listado 2.1 dentro del campo Header estarían los campos de reservas y pasajeros.
    - <Body>: es un elemento obligatorio que contiene información dirigida al destinatario del mensaje. En el ejemplo del Listado 2.1 se puede ver los campos asociados a un itinerario, teniendo este el lugar de partida, de llegada, la fecha de llegada y la preferencia de asiento.
  - <Fault>: es un elemento opcional para notificar errores. En el Listado 2.1 podemos ver que no se encuentra presente, pero en caso de ser utilizado deberá aparecer dentro del elemento *Body* y no puede aparecer más de una vez.

[9] https://www.ibm.com/support/knowledgecenter/es

Listado 2.1: Estructura de un mensaje SOAP

```
<?xml version='1.0' Encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
        <m:reservation xmlns:m="http://travelcompany.example.org/
            reservation"
                env:role="http://www.w3.org/2003/05/soap-envelope/role
                    /next">
            <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:
                reference>
            <m:dateAndTime>2007-11-29T13:20:00.000-05:00</m:dateAndTime>
        </m:reservation>
        <n:passenger xmlns:n="http://mycompany.example.com/employees"
            env:role="http://www.w3.org/2003/05/soap-envelope/role/
                next">
            <n:name>Fred Bloggs</n:name>
        </n:passenger>
    </env:Header>
    <env:Body>
        <p:itinerary xmlns:p="http://travelcompany.example.org/reservation
            /travel">
            <p:departure>
                <p:departing>New York</p:departing>
                <p:arriving>Los Angeles</p:arriving>
                <p:departureDate>2007-12-14</p:departureDate>
                <p:departureTime>late afternoon</p:departureTime>
                <p:seatPreference>aisle</p:seatPreference>
            </p:departure>
            <p:return>
                <p:departing>Los Angeles</p:departing>
                <p:arriving>New York</p:arriving>
                <p:departureDate>2007-12-20</p:departureDate>
                <p:departureTime>mid-morning</p:departureTime>
                <p:seatPreference></p:seatPreference>
            </p:return>
        </p:itinerary>
    </env:Body>
</env:Envelope>
```

- Servicios Web RESTful: RESTful es un protocolo que suele integrar mejor con HTTP que los servicios basado en SOAP, ya que no requieren mensajes XML. Cada petición del cliente debe contener toda la información necesaria para entender la petición, y no puede aprovecharse de ningún contexto almacenado en el servidor.

2.3.2. Arquitectura Servicios Web

Hay que distinguir tres partes fundamentales en los servicios web (**?**):

- El proveedor: es la aplicación que implementa el servicio y lo hace accesible desde Internet.
- El solicitante: cualquier cliente que necesite utilizar el servicio web.
- El publicador: se refiere al repositorio centralizado en el que se encuentra la información de la funcionalidad disponible y como se utiliza.

Por otro lado, los servicios web se componen de varias capas[10]:

[10] https://diego.com.es/introduccion-a-los-web-services

- Descubrimiento del Servicio: responsable de centralizar los servicios web en un directorio común, de esta forma es más sencillo buscar y publicar.
- Descripción del Servicio: como ya hemos comentado con anterioridad, los servicios web se pueden definir a sí mismos, por lo que una vez que los localicemos el Service Description nos dará la información para saber que operaciones soporta y como activarlo.
- Invocación del Servicio: invocar a un Servicio Web implica pasar mensajes entre el cliente y el servidor. Por ejemplo, si utilizamos SOAP *(*Simple Object Access Protocol), el Service Invocation especifica cómo deberíamos formatear los mensajes request para el servidor, y cómo el servidor debería formatear sus mensajes de respuesta.
- Transporte: todos los mensajes han de ser transmitidos de alguna forma entre el servidor y el cliente. El protocolo elegido para ello es HTTP.

2.3.3. Ventajas de los Servicios Web

Las principales ventajas del uso de los servicios web son las siguientes (**?**):

- Permiten la integración "justo-a-tiempo": esto significa que los solicitantes, los proveedores y los agentes actúan en conjunto para crear sistemas que son auto-configurables, adaptativos y robustos.
- Reducen la complejidad por medio del encapsulamiento: un solicitante de servicio no sabe cómo fue implementado el servicio por parte del proveedor, y éste, a su vez, no sabe cómo utiliza el cliente el servicio. Estos detalles se encapsulan en los solicitantes y proveedores. El encapsulamiento es crucial para reducir la complejidad.
- Promueven la interoperabilidad: la interacción entre un proveedor y un solicitante de servicio está diseñada para que sea completamente independiente de la plataforma y el lenguaje.
- Abren la puerta a nuevas oportunidades de negocio: los servicios web facilitan la interacción con socios de negocios, al poder compartir servicios internos con un alto grado de integración.
- Disminuyen el tiempo de desarrollo de las aplicaciones: gracias a la filosofía de orientación a objetos que utilizan, el desarrollo se convierte más bien en una labor de composición.
- Fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

2.3.4. Desventajas de los Servicios Web

El uso de servicios web también tiene algunas desventajas[11]:

- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear.
- Existe poca información de servicios web para algunos lenguajes de programación.
- Dependen de la disponibilidad de servidores y comunicaciones.

2.4. Procesamiento del Lenguaje Natural

El Procesamiento del Lenguaje Natural (PLN) es una rama de la Inteligencia Artificial que se encarga de la comunicación entre máquinas y personas mediante el uso del lenguaje natural (entendiendo como lenguaje natural el idioma usado con fines de comunicación por humanos, ya sea hablado o escrito, como pueden ser el español, el ruso o el inglés). Para ello, una de las tareas principales en el Procesamiento del Lenguaje Natural es interpretar un texto escrito en lenguaje natural y entender su significado, entendiendo como significado la relación entre una palabra o una frase con el mundo. Para realizar dicha acción no solo es necesario el conocimiento del propio lenguaje en que está escrito el texto sino que también es necesario un conocimiento del mundo. Por tanto, uno de los grandes retos del Procesamiento del Lenguaje Natural es la representación del conocimiento. Se deben de buscar técnicas que permitan representar conceptos y relaciones semánticas entre ellos. Una de las principales técnicas de representación en el Procesamiento del Lenguaje Natural son las redes semánticas, en ellas los conceptos que componen el mundo y sus relaciones se representan mediante un grafo. Las redes semánticas se utilizan para representar mapas conceptuales y mentales (**?**). Los nodos están representados por el elemento lingüístico, y la relación entre los nodos sería la arista. Se puede ver un ejemplo en la Figura 2.2, donde el nodo *Oso* representa un concepto, en este caso un sustantivo que identifica a un tipo de animal, y otro nodo *Pelo* el cual también es un sustantivo. La relación entre ambos se ve representada por la arista con valor *tiene*, dando lugar a una característica de este animal: *Oso tiene pelo*.

Existen principalmente tres tipos de redes semánticas (**?**):

- Redes de Marcos: los enlaces de unión de los nodos son parte del propio nodo, es decir, se encuentran organizados jerárquicamente, según un número de criterios estrictos, como por ejemplo la similitud entre nodos. En la Figura 2.3, se muestra un ejemplo de Red de Marco donde los conceptos cama, silla y mesa tienen en común que son objetos.

[11] http://fabioalfarocc.blogspot.com/2012/08/ventajas-y-desventajas-del-soap.html

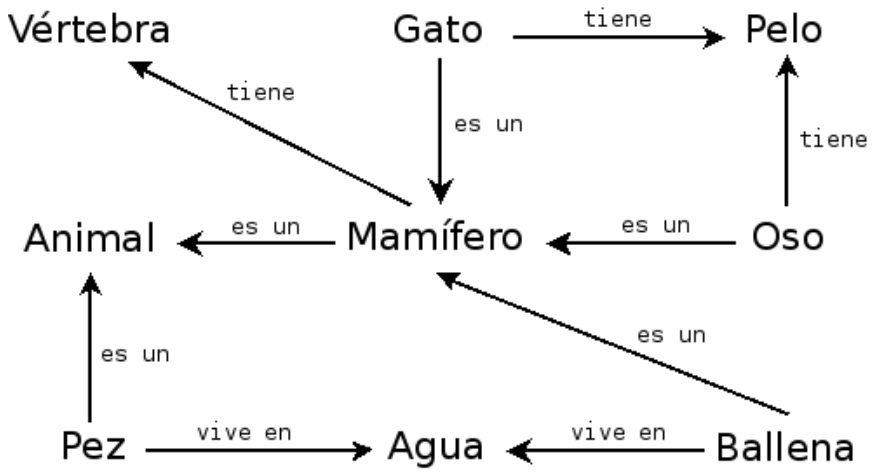

Figura 2.2: Ejemplo de Red Semántica

- Redes IS-A: los enlaces entre los nodos están etiquetados con una relación entre ambos. Es el tipo que habitualmente se utiliza junto con las Redes de Marcos. En la Figura 2.4 se muestra una red IS-A en la que se representa que: el pájaro tiene alas, que es una característica específica de los pájaros.

- Grafos Conceptuales: existen dos tipos de nodos: nodos de conceptos, los cuáles representan una entidad, un estado o un proceso y los nodos de relaciones, que indican como se relacionan los nodos de concepto. En este tipo de red semántica no existen enlaces entre los nodos con una etiqueta, sino que son los propios nodos los que tienen el significado. Se puede ver un ejemplo en la Figura 2.5 (**?**) en la cual la frase *"Man biting dog"* quedaría representada. Los cuadrados implican el concepto y el círculo la relación entre ambos, por lo que en el caso de *man* y *bite*, la acción de morder la realiza *man* siendo éste el agente, y la relación entre *bite* y *dog* sería el objeto.

Para el trabajo que queremos realizar, existen varias aplicaciones web de redes semánticas y son capaces de procesar el Lenguaje Natural. A continuación, hablaremos de algunas de ellas.

2.4.1. ConceptNet

Es una red semántica creada por el MIT *(Massachusetts Institute of Technology)* en 1999, diseñada para ayudar a los ordenadores a entender el significado de las palabras. Está disponible en múltiples idiomas, como el español, el inglés o el chino. ConceptNet ofrece la posibilidad de obtener

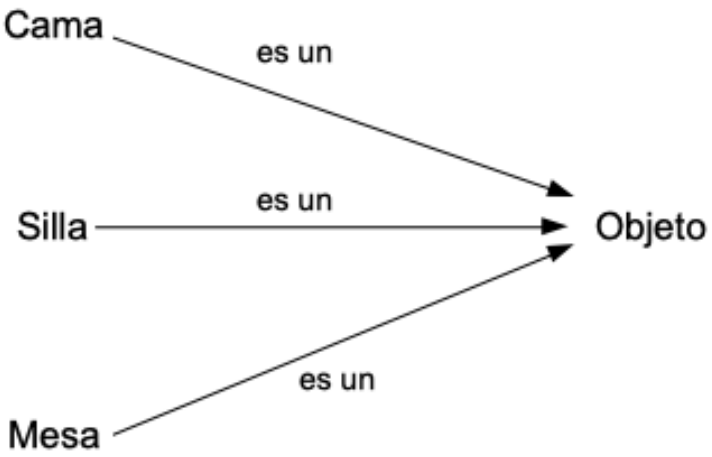

Figura 2.3: Ejemplo de Red de Marco

Figura 2.4: Ejemplo de Red IS-A

de una palabra un listado de sinónimos, términos relacionados, términos derivados, el contexto de la palabra, resultados etimológicamente relacionados, símbolos, etc... Y dispone de una aplicación web[12], donde buscar palabras en distintos idiomas. En la Figura 2.6, se puede ver que devuelve la aplicación ConcepNet para la palabra chaqueta. Por otro lado, ConcepNet dispone de un servicio web[13] que devuelve los resultados en formato JSON. Siguiendo con el mismo ejemplo anterior, se puede ver en el Listado 2.2 los resultados en dicho formato para la palabra chaqueta. Este consta de cuatro campos principales[14]:

[12] http://conceptnet.io/
[13] http://api.conceptnet.io
[14] https://github.com/commonsense/conceptnet5/wiki/AP

Figura 2.5: Ejemplo de Grafo Conceptual

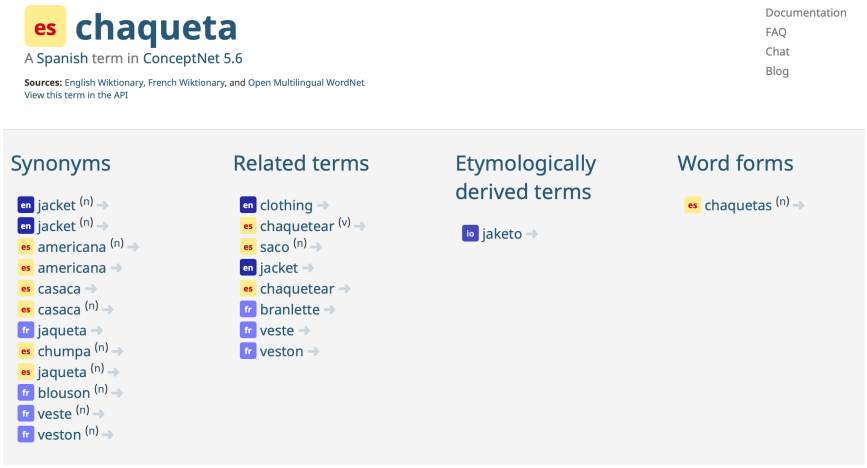

Figura 2.6: Resultados de ConcepNet para la palabra chaqueta

- @context: URL enlazada a un archivo de información del JSON para comprender la API. También puede contener comentarios que pueden ser útiles para el usuario.
- @id: concepto que se ha buscado y su idioma. En nuestro caso, aparece de la siguiente manera: */c/es/chaqueta*, donde *c* significa que es un concepto o término, *es* indica el lenguaje, en este caso, el español y por último *chaqueta* que es la palabra buscada.
- edges: representa una estructura de datos devueltos por Conceptnet compuesta por:
  - @id: describe el tipo de relación que existe entre la palabra introducida y la devuelta. En el Listado 2.2 se indica que la palabra *americana* es un sinónimo de *chaqueta*.
  - @type: define el tipo del id, es decir, si es una relación (edge) o un término (nodo).
  - dataset: URI que representa el conjunto de datos creado.
  - end: nodo destino, que a su vez se compone de:
    - @id: coincide con la palabra del id anterior.

- @type: define el tipo de id, como se ha explicado anteriormente.
- label: puede ser la misma palabra buscada o una frase más completa, donde adquiera significado la palabra obtenida.
- language: lenguaje en el que está la palabra devuelta de la consulta.
- term: enlace a una versión mas general del propio término. Normalmente, suele coincidir con la URI.

- license: aporta información sobre como debe usarse la información proporcionada por conceptnet.
- rel: describe la relación que hay entre la palabra origen y destino, dentro del cual hay tres campos: @id, @type y label, descritos anteriormente.
- sources: indica porqué ConceptNet guarda esa información, este campo como los anteriores, es un objeto que tiene su propio id y un campo @type, A parte, hay un campo *contributor*, en el que aparece la fuente por la que se ha obtenido ese resultado y por último un campo *process* indicando si la palabra se ha añadido mediante un proceso automático.
- start: describe el nodo origen, es decir, la palabra que hemos introducido en ConceptNet para que haga la consulta, este campo esta compuesto por elementos ya descritos como son: @id, @type, label, language y term.
- surfaceText: algunos datos de ConceptNet se extraen de texto en lenguaje natural. El valor de surface text muestra lo que era este texto, puede que este campo tenga valor nulo.
- weight: indica la fiabilidad de la información guardada en conceptnet, siendo normal que su valor sea 1.0. Cuanto mayor sea este valor, más fiables serán.

- view: describe la longitud de la lista de paginación, es un objeto con un id propio, y además, aparecen los campos *firstPage* que tiene como valor un enlace a la primera pagina de los resultados obtenidos, y *nextPage* que tiene un enlace a la siguiente página de la lista.

Listado 2.2: JSON devuelto por la API de ConceptNet para la palabra chaqueta


```
{
   "@context": [
      "http://api.conceptnet.io/ld/conceptnet5.6/context.ld.json"
],
   "@id": "/c/es/chaqueta",
   "edges": [
{
```

```
    "@id": "/a/[/r/Synonym/,/c/es/chaqueta/n/,/c/es/americana/]",
    "@type": "Edge",
    "dataset": "/d/wiktionary/fr",
    "end": {
    "@id": "/c/es/americana",
    "@type": "Node",
    "label": "americana",
    "language": "es",
    "term": "/c/es/americana"
},
    "license": "cc:by-sa/4.0",
    "rel": {
    "@id": "/r/Synonym",
    "@type": "Relation",
    "label": "Synonym"
},
    "sources": [
{
    "@id": "/and/[/s/process/wikiparsec/1/,/s/resource/wiktionary/fr/]",
    "@type": "Source",
    "contributor": "/s/resource/wiktionary/fr",
    "process": "/s/process/wikiparsec/1"
}
],
    "start": {
    "@id": "/c/es/chaqueta/n",
    "@type": "Node",
    "label": "chaqueta",
    "language": "es",
    "sense_label": "n",
    "term": "/c/es/chaqueta"
},
    "surfaceText": null,
    "weight": 1.0
},

]

    "view": {
    "@id": "/c/es/chaqueta?offset=0&limit=20",
    "@type": "PartialCollectionView",
    "comment": "There are more results. Follow the 'nextPage' link for more.",
    "firstPage": "/c/es/chaqueta?offset=0&limit=20",
    "nextPage": "/c/es/chaqueta?offset=20&limit=20",
    "paginatedProperty": "edges"
    }

}
```


2.4.2. Thesaurus

Es una aplicación web[15] que se autodefine como el principal diccionario de sinónimos de la web. Esta página ofrece la posibilidad de introducir una palabra para poder conocer sus sinónimos, pero solamente devuelve resultados en inglés. Aparte del listado de sinónimos, Thesaurus indica que tipo de palabra es y una definición de la misma así como un listado de antónimos y un listado de palabras relacionadas con dicho concepto. Por otro lado esta aplicación proporciona una API[16] tipo RESTful que obtiene los sinónimos de una palabra mediante una petición HTTP GET a la url *http://thesaurus.altervista.org/thesaurus/v1*. Este devuelve los resultados en formato XML o JSON. El contenido de la respuesta es una lista y cada elemento de esta lista contiene un par de elementos: categoría y sinónimos. Este último a su vez contiene una lista de sinónimos separados por el carácter |. Se puede ver en el Listado 2.3 un ejemplo para la palabra *peace* de como

[15] https://www.thesaurus.com/
[16] http://thesaurus.altervista.org/

sería el resultado de una petición en formato XML y en el Listado 2.4 para la misma palabra, *peace*, en formato JSON. Ambos son muy similares, por ejemplo en formato XML se puede ver que devuelve el tipo de categoría de las palabras, en este caso son sustantivos y a continuación aparcen los sinónimos. En caso de que alguna palabra sea un antónimo aparecera entre paréntesis al lado de la misma, como ocurre con la palabra *war*. Por otro lado, el formato JSON devuelve dentro del campo *category / categoría* todos los sinónimos, y en caso de ser un antónimo aparecerá de la misma forma que en el formato XML.

Listado 2.3: Ejemplo de salida de Thesaurus en formato XML para la palabra *peace*

```
<response>
   <list>
      <category>(noun)</category>
      <synonyms> order|war (antonym) </synonyms>
   </list>
   <list>
      <category>(noun)</category>
      <synonyms> harmony|concord|concordance </synonyms>
   </list>
   <list>
      <category>(noun)</category>
      <synonyms> public security|security </synonyms>
   </list>
   <list>
      <category>(noun)</category>
      <synonyms> peace treaty|pacification|treaty|pact|accord </synonyms>
   </list>
```

Listado 2.4: Ejemplo de salida de Thesaurus en formato JSON para la palabra *peace*

```
{
   "response":
   [
      { "list":
         { "category": "(noun)", "synonyms": "order|war (antonym)" }
      },
      { "list":
         { "category": "(noun)", "synonyms": "harmony|concord|concordance" }
      },
      { "lista":
         { "categoria": "(noun)", "synonyms": "public security|security" }
      },
      { "lista":
         { "categoria": "(noun)", "synonyms": "peace treaty|pacification|treaty|
              pact|accord" }
      }

   ]
}
```

2.4.3. Thesaurus Rex

Thesaurus Rex[17] es una red semántica que solo admite palabras en inglés y que permite obtener las palabras relacionadas, con una palabra o las categorías que comparten dos palabras, por ejemplo si se introducen las

[17] http://ngrams.ucd.ie/therex3/

palabras *coffe y cola*, las categorías que comparten dichos conceptos son *cold-beverage, dark-beverage, stimulating-beverage, etc...* Si por el contrario, únicamente se ha introducido una palabra como se puede ver en la Figura 2.7 para la palabra *house*. La aplicación devuelve un listado de las categorías más utilizadas por los hablantes de dicha lengua, como por ejemplo *permanent-structure, inanimate-object, everyday-object, etc...*, otro listado de las categorías matizadas de dicho concepto, es decir, atributos del concepto buscado, como por ejemplo *permanent, fixed, wooden, etc...* y por último un listado de categorías simples del concepto, como por ejemplo *structure, object, item, building, etc...*

Por otro lado, la aplicación devuelve estos resultados en formato XML, y este como se puede ver en el Listado 2.5 para la palabra *house*, los divide en distintos campos: *Categories*, *Modifiers* y *CategoryHeads*. Todos los resultados tienen un peso *(weight)* asignado, esto significa que cuanto mayor sea el peso mayor es la similitud con el concepto dado, y en la página aparecerá dicha palabra en un tamaño superior al resto. Los campos que se encuentran dentro del apartado *categories* son los resultados más utilizados en ese momento por los hablantes y que Thesaurus Rex ha encontrado, como por ejemplo *permanent-structure*, los que se encuentran dentro de *modifiers*, como se ha comentado anteriormente son atributos del concepto a buscar, como por ejemplo *fixed* y por último los que se encuentran en *categoryHeads* son las categorías más simples que se han encontrado para dicho concepto, como por ejemplo *structure*. Thesaurus Rex utiliza la Web para generar sus resultados, con lo cual la información disponible no es fija, sino que varía según los datos actuales de la web. La ventaja de utilizar esta herramienta es que se encuentra en continua actualización, pero el inconveniente es que en algunos casos la información puede resultar un poco extraña dado que se crea semiautomáticamente desde contenido de la web (**?**).

Listado 2.5: Ejemplo formatos XML Thesaurus Rex para la palabra *house*

```
<MemberData>
   <Categories kw="house">
      <Category weight="91"> large:object </Category>
      <Category weight="307"> inanimate:object </Category>
      <Category weight="261"> everyday:object </Category>
      <Category weight="154"> sensitive:area </Category>
      <Category weight="318"> permanent:structure </Category>
      <Category weight="194"> permanent:construction </Category>
      <Category weight="148"> permanent:installation </Category>
      <Category weight="98"> fixed:object </Category>
   </Categories>

   <Modifiers kw="house">
      <Modifier weight="8"> recognizable </Modifier>
      <Modifier weight="9"> relevant </Modifier>
      <Modifier weight="863"> permanent </Modifier>
      <Modifier weight="15"> moderate </Modifier>
      <Modifier weight="477"> fixed </Modifier>
      <Modifier weight="5"> odd </Modifier>
      <Modifier weight="7"> archaeological </Modifier>
      <Modifier weight="5"> electrical </Modifier>
   </Modifiers>

   <CategoryHeads kw="house">
      <CategoryHead weight="6"> protection </CategoryHead>
```

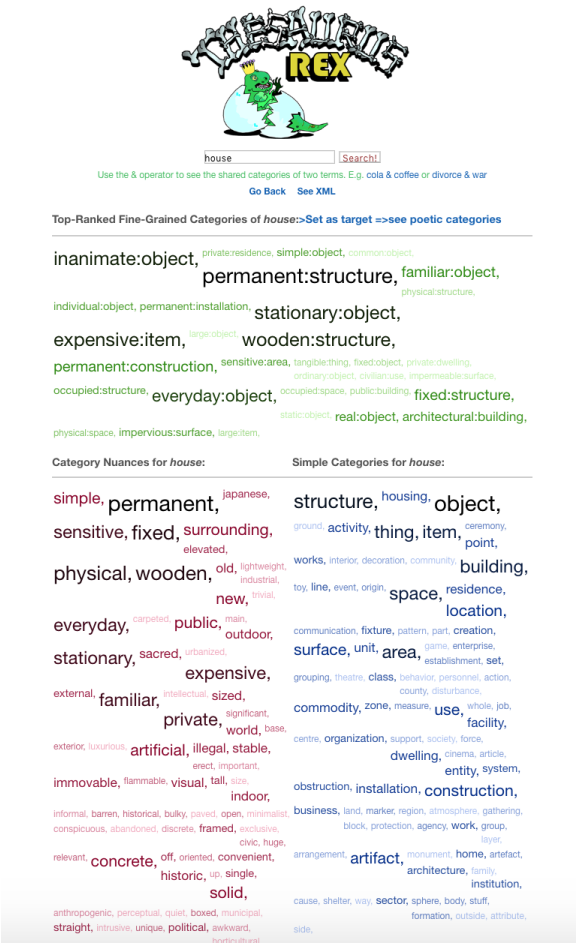

Figura 2.7: Resultados búsqueda Thesaurus Rex con la palabra *house*

```
        <CategoryHead weight="40"> obstruction </CategoryHead>
        <CategoryHead weight="5"> whole </CategoryHead>
        <CategoryHead weight="3320"> object </CategoryHead>
        <CategoryHead weight="2340"> structure </CategoryHead>
        <CategoryHead weight="98"> commodity </CategoryHead>
        <CategoryHead weight="713"> thing </CategoryHead>
        <CategoryHead weight="2"> theatre </CategoryHead>
    </CategoryHeads>
</MemberData>
```

2.4.4. Metaphor Magnet

Metaphor Magnet es una aplicación web[18], que crea metáforas a partir de una palabra y que solo está disponible para el inglés. El objetivo de Metaphor Magnet (**?**) es encontrar y explotar metáforas comunes en textos

[18] http://ngrams.ucd.ie/metaphor-magnet-acl/

cotidianos (en este caso, en los n-gramas de Google) y usar estos mapeos para interpretar metáforas. Se entiende como n-grama (**?**) a una subsecuencia de n elementos consecutivos en una secuencia dada y estos pueden ser bigramas, tigramas, etc... Por ejemplo, en el texto "Platero y yo"[19], si se toma como elementos los caracteres que lo componen, sus trigramas serían: Pla, lat, ate, ter, ero, ro-, o-y, -y-, y-y, -yo[20]. Para el caso de "El horizonte es límite de lo que podemos ver", si se establecen como elementos a las palabras del texto, sus bigramas son: El horizonte, horizonte es, es límite, límite de, de lo, lo que, que podemos, podemos ver.

La propia aplicación permite introducir palabras o frases, a estas últimas se les puede añadir el signo + seguido de la palabra para indicar que se busquen resultados enfocados a la positividad y con un - indicando lo inverso. Por ejemplo, en la Figura 2.8 se puede ver un ejemplo introduciendo la frase *"life is a +game"* añadiendo el signo + delante de la palabra game, obteniendo como resultado dos columnas, la columna del lado izquierdo (*Target Metaphors*) se refiere a metáforas enfocadas al concepto *"live"* y la columna del lado derecho (*Source Metaphors*) devuelve metáforas positivas sobre el concepto *game*. Por otro lado, en el caso de añadir una única palabra como en lo mostrado en la Figura 2.9 donde se puede ver el resultado obtenido para la palabra *house*, devuelve métaforas propias del concepto, por ejemplo *protecting:home*. Esta consulta devuelve un fichero XML como el expuesto en el Listado 2.6, donde aparece la etiqueta *<Source Name>* seguido de la palabra a buscar. Otra etiqueta *<Score>* que muestra un número, cuanto mayor sea este indica que ese resultado es más acertado respecto al concepto introducido y se mostrará en un tamaño mayor como por ejemplo *"tall:building"* con una etiqueta *score* de 86. Y por último, una etiqueta *<Text>* con la metáfora, por ejemplo *"protecting:home"*.

Listado 2.6: Ejemplo formatos XML Metaphor Magnet para la palabra *house*

```
<Metaphor>
   <Source Name="house">
      <Text> towering:mountain </Text>
      <Score> 88 </Score>
   </Source>
   <Source Name="house">
      <Text> protecting:home </Text>
      <Score> 86 </Score>
   </Source>
   <Source Name="house">
      <Text> tall:building </Text>
      <Score> 86 </Score>
   </Source>
   <Source Name="house">
      <Text> charming:castle </Text>
      <Score> 85 </Score>
   </Source>
   <Source Name="house">
      <Text> beautiful:tree </Text>
      <Score> 84 </Score>
   </Source>
   <Source Name="house">
```

[19] https://www.ecured.cu/N-grama

[20] Se ha añadido el símbolo '-'como indicador de espacio en blanco

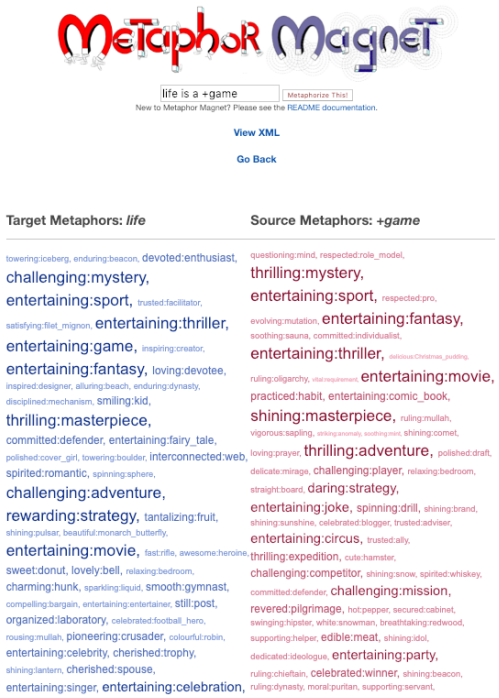

Figura 2.8: Resultados búsqueda Metaphor Magnet con la frase *life is a game*

```
        <Text> charming:mansion </Text>
        <Score> 83 </Score>
    </Source>
    <Source Name="house">
        <Text> strong:rock </Text>
        <Score> 80 </Score>
    </Source>
    <Source Name="house">
        <Text> strong:elephant </Text>
        <Score> 80 </Score>
    </Source>
</Metaphor>
```

2.4.5. WordNet

WordNet es un *corpus*[21] perteneciente a NLTK *(Natural Languaje Toolkit)*[22] que almacena distintos tipos de palabras como sustantivos, verbos, adjetivos y adverbios ignorando preposiciones, determinantes y otras palabras funcionales en varios idiomas como el español, el inglés o el francés. Los conceptos se agrupan en conjuntos de sinónimos cognitivos llamados *synsets*. Cada *synset* contiene:

- Hiperónimos, los cuales son palabras cuyo significado está incluido en

[21] Colección de documentos de texto
[22] Conjunto de bibliotecas y programas para el Procesamiento del Lenguaje Natural

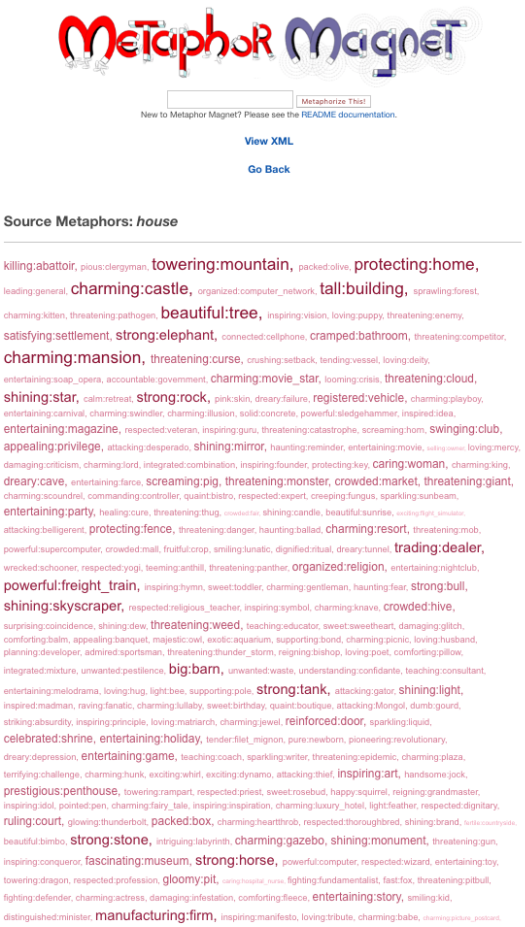

Figura 2.9: Resultados búsqueda Metaphor Magnet con la palabra *house*

el de otras[23], por ejemplo: mamífero es hiperónimo de gato y de perro ya que los gatos y los perros pertenecen al conjunto de los mamíferos.

- Hipónimos, son palabras cuyo significado incluyen el de otra[24], por ejemplo: gato es hipónimo de mamífero ya que está incluido dentro del conjunto de los mamíferos.

- Holónimos, son palabras que representan el todo respecto a una parte, por ejemplo: coche es el holónimo de rueda, volante y acelerador ya que forman parte de un todo, que es el coche.

- Por último, contiene una breve definición y en muchas ocasiones, oraciones cortas que explican su significado.

[23] https://dle.rae.es/?id=KRW1qe2
[24] https://dle.rae.es/?id=KU5UAn5

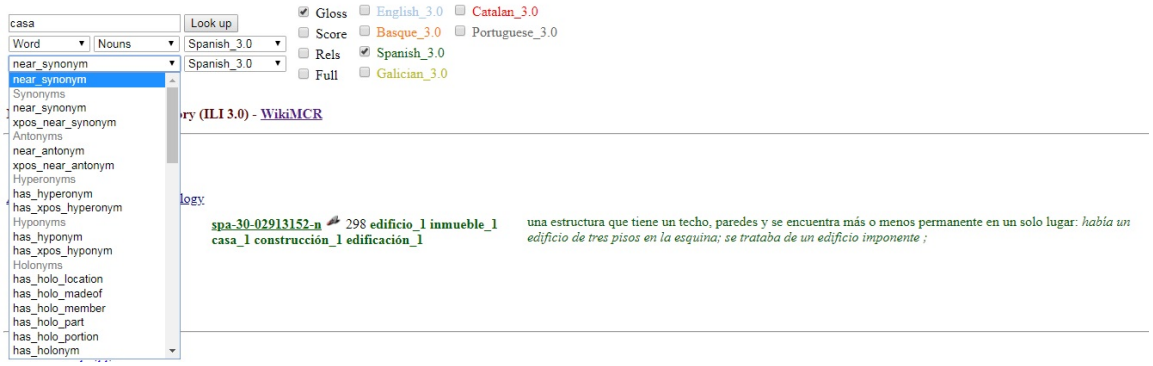

Figura 2.10: Resultados de la búsqueda en EuroWordNet para la palabra *casa*

Además, una vez obtenidos los sinónimos de una palabra, se pueden obtener para cada uno de estos una lista de antónimos. En caso de que la palabra tenga distintos significados, aparecerá un *synset* por cada una[25].

Existen varias aplicaciones web que implementan este *corpus*, una de las más completas es EuroWordNet, ya que está disponible en varios idiomas y permite extraer sinónimos, antónimos, hiperónimos, hipónimos y holónimos. Además de oraciones de ejemplo, y definiciones de cada uno de los *synsets* obtenidos.

.

En la Figura 2.10 aparecen los sinónimos correspondientes a uno de los *synset* devueltos cuando buscamos la palabra "casa" en EuroWordNet. Los datos obtenidos son el *offset* del *synset*(en este caso: spa-30-02913152-n), que es un código que lo identifica inequívocamente y que finaliza con una letra que describe la categoría gramatical de los sinónimos devueltos, en este caso la "n", que referencia a *noun*, que significa nombre en inglés. A continuación, una serie de sinónimos como por ejemplo edificio o inmueble. Aparte, devuelve una definición para contextualizar el concepto, en este caso: una estructura que tiene un techo, paredes y se encuentra más o menos permanente en un solo lugar. Y un ejemplo para que en el caso de que la definición no sea suficientemente clara para el usuario, le ayude a comprenderlo, en el caso de la palabra casa el ejemplo devuelto es: había un edificio de tres pisos en la esquina. Además, en el desplegable de la izquierda se muestran los tipos que se pueden buscar de dicho concepto. Por ejemplo, si seleccionamos *has hyperonym* se obtendrán los hiperónimos de casa, o seleccionando *has hyponym* se obtendrán los hipónimos.

[25] http://www.nltk.org/howto/wordnet.html

Capítulo 3

Herramientas Utilizadas

En este capítulo se van a explicar las herramientas utilizadas para el desarrollo de este trabajo, en el apartado 3.1 se explicará Django que es el *framework* utilizado para el desarrollo del servicio web y en el apartado 3.2 se explicará SpaCy, que es la herramienta que se utilizó para la clasificación de palabras. Las herramientas y recursos que se utilizaron para la gestión del proyecto se explicarán en un capítulo aparte.

3.1. Django

Django es un *framework* de alto nivel que permite el desarrollo rápido de sitios web seguros y mantenibles y se basa en el patrón MVC[1]. Fue desarrollado entre los años 2003 y 2005 por un grupo de programadores que se encargaban de crear y mantener sitios web de periódicos. Es gratuito y de código abierto y dispone de una gran documentación actualizada así como muchas opciones de soporte gratuito y de pago.

Algunas de las razones por las que se ha elegido este *framework*han sido las siguientes: [2]

- Seguridad: Implementa por defecto algunas medidas de seguridad para evitar SQL Injection, Cross site request forgery (CSRF) o Clickjacking por JavaScript.
- Escalabilidad: Se puede pasar de una aplicación sencilla a otra más compleja rápidamente, ya que es muy fácil añadir nuevos módulos al *framework*.
- Fácil acceso a bases de datos: Mediante ORM, que es su interfaz para

[1] https://docs.djangoproject.com/en/2.0/
[2] https://openwebinars.net/blog/que-es-django-y-por-que-usarlo/

TEXT	LEMMA	POS	TAG	DEP	SHAPE	ALPHA	STOP
Apple	apple	PROPN	NNP	nsubj	Xxxxx	True	False
is	be	VERB	VBZ	aux	xx	True	True
looking	look	VERB	VBG	ROOT	xxxx	True	False
at	at	ADP	IN	prep	xx	True	True
buying	buy	VERB	VBG	pcomp	xxxx	True	False
U.K.	u.k.	PROPN	NNP	compound	X.X.	False	False

Figura 3.1: Ejemplo de clasificación de palabras

el acceso a bases de datos, se pueden hacer consultas de manera muy intuitiva.

Además, es muy popular por lo que para resolver cualquier problema que surja, se puede encontrar la solución consultando la documentación o preguntando en algún foro.

3.2. SpaCy

SpaCy es una biblioteca de código abierto para el Procesamiento del Lenguaje Natural en Python, soporta más de 34 idiomas entre ellos el español. Se requería una herramienta que clasificara según su categoría gramatical las 1000 palabras más utilizadas del castellano, ya que solo se necesitaban los verbos, pronombres sustantivos y adverbios. Se pensó inicialmente en la biblioteca NLTK de Python, pero tras una prueba inicial se vio que la clasificación no era una fiable por lo que se descartó su uso y se buscaron otras opciones, se probó con SpaCy y el resultado fue muy satisfactorio, por lo se decidió utilizar en el proyecto para el fin anteriormente descrito.

SpaCy según su página web [3], tiene el mejor índice de acierto como analizador sintáctico, para su utilización hay que importar la biblioteca de idioma correspondiente (en este caso español: "es_core_news_sm") y pasar como parámetro las palabras que se deseen clasificar, el resultado será una serie de etiquetas tal y como se pueden apreciar en la tabla 3.1, que es un ejemplo de las etiquetas generadas para una frase en lengua inglesa y que se explica a continuación:

- La columna *text* indica cual es la palabra que se ha procesado. En el caso de la primera fila es la palabra *Apple*.
- La columna *lemma* indica la forma base de la palabra procesada. En la primera fila es la palabra *apple* en minúsculas.
- La columna *pos* es la etiqueta asignada a dicha palabra. Indicando lo siguiente:

[3] https://spacy.io/

  - PROPN: Nombre propio. Por ejemplo *Apple*
  - VERB: Verbo. Como *is* que es una forma del verbo *to be*.
  - ADP: Preposicion. Por ejemplo *at*.

- La columna *tag* indica cual es la palabra que se ha procesado con más detalle.
- La columna *dep* indica la dependencia sintáctica de la palabra en la frase. Por ejemplo en la segunda frase *is* es un verbo auxiliar.
- La columna *shape* indica la apariencia de la palabra procesada, es decir, si está en mayúsculas o si tiene algún signo de puntuación. Por ejemplo en la última columna, la palabra *U.K.* tiene un *shape* X.X. ya que está formado por dos letras mayúsculas y dos puntos.
- La columna *alpha* es un valor booleano que tendrá el valor *True* si la palabra es un carácter alfanumérico y *False* si no lo és. Por ejemplo: *buying* tiene la etiqueta *alpha* a *True* ya que está formada por caracteres alfanuméricos y *U.K.* la tiene a *False* ya que los "." no se consideran caracteres alfanuméricos.
- La columna *stop* es un valor booleano que tendrá el valor *True* si la palabra forma parte de las palabras más comunes del lenguaje en el que se encuentra *False* si no lo és. Por ejemplo: *is* que es una forma del verbo *to be* tiene esta etiqueta a *True* ya que es muy utilizado en el lenguaje inglés.

Capítulo 4

Gestión del Proyecto

Desde el inicio de la realización del proyecto, se han seguido ciertas pautas para que el funcionamiento de este fuese lo más eficaz posible, es por ello que se han tenido reuniones asiduamente con los directores de este trabajo, cada dos/tres semanas en donde se corregían los fallos, se indicaban las siguientes tareas por hacer (tanto de código como de memoria) y se buscaban soluciones a los problemas y dudas que pudieran surgir o plantearse. Por otro lado, ha habido una comunicación con ambos directores vía email para pequeñas dudas o para concretar citas de tutorías, no siendo estas las reuniones programadas, sino como un plus a la hora de realizar el proyecto. En relación con el código de este trabajo, para poder llevar un control de las versiones del mismo y de la versión del código de cada integrante se ha utilizado la plataforma de desarrollo online GitHub. GitHub

GitHub es una plataforma de desarrollo colaborativo online que utiliza el sistema de control de versiones distribuido Git[1] en el que cada miembro del equipo de desarrollo tiene su propia copia del repositorio online en local. Los cambios que realice cada usuario se guardan en esta copia local y cuando desee se suben al repositorio online mediante la acción *commit* y *push*. En caso de que haya algún conflicto, el sistema te permite decidir como gestionarlos. Las características principales a destacar de GitHub son:

- Licencias: cuando creamos un proyecto, GitHub nos permite añadir un archivo de texto en el cuál se puede explicar el tipo de licencia que dispone el proyecto. Es por ello, que GitHub tiene diferentes plantillas de licencias como por ejemplo para una licencia tipo GPL, MIT o Apache. Estas licencias determinarán el uso que el usuario puede hacer de los proyectos y si se tiene que realizar diferentes acciones como mencionar el proyecto original o al desarrollador del proyecto original.
- Gráficas: GitHub dispone de una pestaña *Insights*, donde se puede ver

[1] https://github.com/

en forma de gráfica las contribuciones (*commits*) que ha realizado cada integrante, es decir, las líneas de código tanto añadido como eliminado. También se puede saber que día se realizaron los *commits* y cuantos.

- Red Social: el usuario dispone de perfil y se pueden buscar a otros usuarios, dando la posibilidad de poder seguirse.

- Wiki: cada proyecto puede tener su propia wiki con manuales e información relativas a éste.

GitHub también dispone de una pestaña *Issues*, donde los usuarios pueden escribir las tareas que hay que realizar, haciendo éste el símil de una pizarra o tablero, y de esta forma poder tener una gestión de tareas y saber cada integrante del proyecto en que proceso se encuentra cada tarea. Nuestro equipo no se rige por una metodología concreta pero si que hemos adoptado ciertas características de las metodologías ágiles para realizar nuestro trabajo. Hemos hecho uso de un gestor de tareas para emplearlo como radiador de información y que así todos los integrantes del proyecto puedan conocer en cada momento el estado de este. Podríamos hacer utilizado, como hemos comentado anteriormente el tablero de GitHub, pero nos hemos decantado por Trello, ya que dispone de una interfaz simple, amigable y que no lleva a confusión a la hora de crear nuevas tareas o moverse por el tablero. Existen dos tipos de tareas: las relacionadas con código y las relacionadas con la memoria. Se ha realizado una distinción entre ambas, ya que la forma de cambiar su estado en el tablero varía significativamente. La forma de poder distinguir estas, es que delante de la descripción de la tarea aparecerá la palabra CÓDIGO o la palabra MEMORIA. Por otro lado se han añadido tres columnas:

- Lista de tareas: en dónde se ven representadas todas las tareas a realizar, desgranadas al mayor detalle posible e intentando que éstas sean lo más independientes las unas de las otras. De esta forma, nos aseguramos que cada integrante del equipo trabaja en una tarea específica que no influye en el trabajo del compañero.

- En proceso: en el momento en el que un integrante del grupo se asigna una tarea, esta se pasa de la columna Lista de Tareas a esta. Lo que indica que se encuentra en proceso de realización y que ningún otro compañero puede realizarla.

- Hecho: cuando una tarea se encuentra en dicha columna implica que la tarea ha sido terminada y validada, esta validación depende del tipo de tarea que sea, si es de tipo código la validación la deben hacer los desarrolladores y si es de tipo memoria la deben hacer los directores.

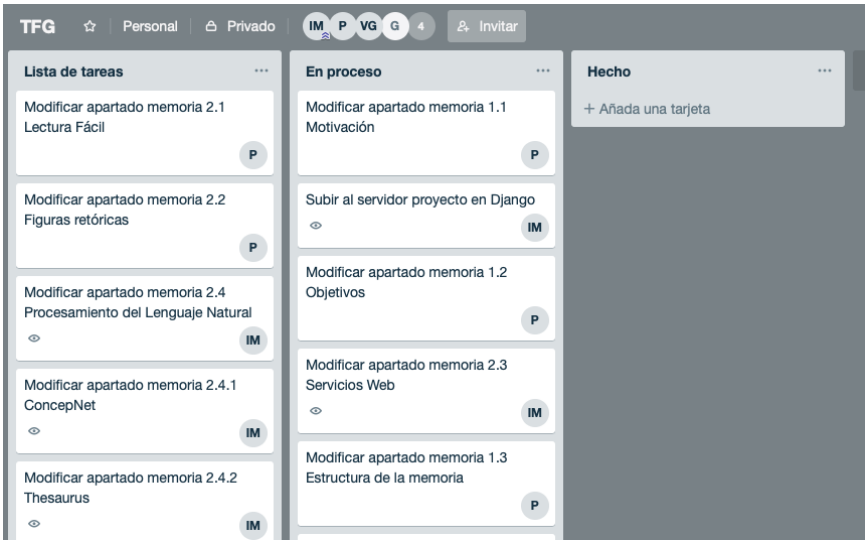

Figura 4.1: Ejemplo Gestor de Tareas en Trello

Para que se pueda entender con mayor claridad, podemos ver la Figura 4.1 dónde en cada columna se ve claramente la descripción de la tarea y quién la tiene asignada.

Capítulo 5

Servicios Web Implementados

5.1. Servicio Web Para ConcepNet

5.2. Servicio Web para WordNet

Para la implementación y desarrollo del servicio web, se ha hecho uso de MCR (*Multilingual Central Repositoy*), el cuál es una base de datos de código abierto que integra distintas versiones de WordNet para seis lenguajes diferentes: Inglés, Español, Catalán, Vasco, Gallego y Portugues.

Capítulo 6

Trabajo Realizado

En este capítulo vamos a describir que trabajo hemos hecho cada uno

6.1. Trabajo realizado por Irene

Primero investigué las bibliotecas que utilizaremos para el procesado de las palabras, al principio encontramos una biblioteca para el procesado de texto en Python, que es la ntlk pero vimos que las etiquetas que ponía a las palabras no eran del todo correctas por lo que buscamos otra biblioteca y encontramos Spacy, con esta ya pudimos etiquetar bien todas las palabras diseñando un programa inicialmente en el Jupyter. A continuación, investigué que tecnologías utilizar para la realización del prototipo tecnológico, encontramos como entorno de desarrollo Pycharm y como framework Django. Una vez seleccionadas las tecnologías, investigamos como se utilizaban y nos pusimos a trabajar en el prototipo tecnológico.

Yo me encargué de conectar las vistas html con la lógica en Python, a continuación vimos como se implementaba un formulario y como se hacia una redirección a vista. Cuando supimos como se hacia todo esto, integramos el código desarrollado en Jupyter en nuestro servicio web finalizando el prototipo tecnológico.

En cuanto a la memoria me la dividí a partes iguales con Pablo, intentando que los dos toquemos todo, por lo que ambos redactamos tanto una parte de la introducción (en la que redacté la motivación) como el estado de la cuestión (yo hice el apartado de lectura fácil y Procesamiento del Lenguaje Natural).

La investigación de como funcionaba Conceptnet y su API la hicimos conjuntamente.

6.2. Trabajo realizado por Pablo

Al igual que mi compañera, lo primero que hicimos fue investigar como podíamos etiquetar las palabras, encontramos la librería ntlk de Python para hacerlo, pero tras un primer intento nos dimos cuenta de que muchas palabras no estaban etiquetadas como deberían por lo que decidimos buscar alternativas, indagando un poco encontramos Spacy, la probamos y obtuvimos unos resultados mucho mejores que con ntlk por lo que decidimos utilizar esta última (todo esto lo hicimos desde el Jupyter).

Cuando terminamos de etiquetar las palabras nos pusimos a investigar herramientas para el desarrollo del prototipo tecnológico y nos decantamos por utilizar Django como framework integrado en Pycharm, que es el entorno de desarrollo.

A continuación empezamos el desarrollo del prototipo tecnológico primero investigando como se utilizaban estas tecnologías(implementar formularios, hacer las redirecciones a vista...). Para finalizar migramos lo hecho desde Jupyter a nuestro servicio web.

Irene y yo nos dividimos la redacción de la memoria de tal manera que los dos hicimos tanto la parte de la introducción como del estado de la cuestión, de la introducción a mí me tocó la parte de los objetivos y del estado de la cuestión la parte de figuras retóricas y servicios web.

La investigación de como funcionaba Conceptnet y su API la hicimos de manera conjunta.

El diseño de la Interfaz debe cumplir las Ocho Reglas de Oro del diseño de interfaces:

- Consistencia: La funcionalidad debe ser similar a otras aplicaciones que el usuario está acostumbrado a utilizar. En cuanto a la interfaz debe tener los mismos colores, iconos, formas, botones, mensajes de aviso... Por ejemplo, si el usuario está acostumbrado a que el botón de eliminar o cancelar sea rojo, no debemos añadirle uno de color verde.

- Usabilidad Universal: Debemos tener en cuenta las necesidades de los distintos tipos de usuario, como por ejemplo, atajos de teclado para un usuario experto o filtros de color para usuarios con deficiencias visuales.

- Retroalimentación activa: Por cada acción debe existir una retroalimentación legible y razonable por parte de la aplicación. Por ejemplo, si el usuario quiere guardar los datos obtenidos de la búsqueda, la aplicación debe informarle de si han sido guardados o no.

- Diálogos para conducir la finalización: El usuario debe saber en que paso se encuentra en cada momento. Por ejemplo, en un proceso de

compra que conlleva varios pasos hasta la finalización de la misma, se le debe informar donde se encuentra y cuánto le queda para terminar.

- Prevención de errores: La interfaz debe ayudar al usuario a no cometer errores serios, y en caso de cometerlos se le debe dar una solución lo más clara y sencilla posible. Por ejemplo, deshabilitando opciones o indicando en un formulario el campo en el cual se ha producido el error sin perder la información ya escrita.

- Deshacer acciones fácilmente: Se debe dar al usuario la capacidad de poder deshacer o revertir acciones de una manera sencilla.

- Sensación de control: Hay que dar al usuario la sensación de que tiene en todo momento el control de la aplicación, añadiendo contenidos fáciles de encontrar y de esta forma no causarle ansiedad o frustración por utilizar nuestra aplicación.

- Reducir la carga de memoria a corto plazo: La interfaz debe ser lo más sencilla posible y con una jerarquía de información evidente, es decir, hay que minimizar la cantidad de elementos a memorizar por el usuario.

Capítulo 7

Conclusiones y Trabajo Futuro

Conclusiones del trabajo y líneas de trabajo futuro.

Chapter 7

Conclusions and Future Work

Conclusions and future lines of work.

Apéndice A

Título

Contenido del apéndice

Apéndice B

Título

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes