Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

INDUSTRIA INGENIARITZA TEKNIKOKO ATALA

SECCIÓN INGENIERÍA TÉCNICA INDUSTRIAL

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

2016 / 2017

MINERÍA DE CONTENIDO WEB MÉDICO

Memoria del Proyecto

DATOS DE LA ALUMNA O DEL ALUMNO	DATOS DEL DIRECTOR O DE LA DIRECTORA
NOMBRE **CARLOS**	NOMBRE **KOLDOBIKA // AITZIBER**
APELLIDOS **SÁNCHEZ CORONAS**	APELLIDOS **GOJENOLA GALLETEBEITIA // ATUTXA SALAZAR**
	DEPARTAMENTO **Sistemas y lenguajes**
FDO.:	FDO.:
FECHA: **13-06-2017**	FECHA: **13-06-2017**

Índice General

1. Introducción 1

2. Planteamiento inicial 2

2.1. Objetivos 2

2.1.1 Objetivos principales 2

2.1.2 Objetivos personales 2

2.2. Arquitectura 2

2.3. Alcance del proyecto 3

2.3.1. Ciclo de vida 3

2.3.2. Prototipos 4

2.4. Herramientas 5

2.4. Planificación temporal 7

2.4.1. Etapas 7

2.4.2. Estructura de Desglose de Trabajo (EDT) 7

2.4.3. Descripción de tareas 9

2.4.4. Diagrama de Gantt Inicial 11

2.4.5. Diagrama de Gantt Final 18

2.5. Gestión de riesgos 18

3. Estado del arte 21

3.1. Ontología 21

3.2. OWL 21

3.3. SPARQL 22

4. Captura de requisitos 23

4.1. Jerarquía de actores 23

4.2. Diagrama de casos de uso 23

5. Desarrollo de prototipos 24

5.1. Prototipo 1: Captura y preproceso de información de internet 24

5.1.1 Análisis y diseño: 24

5.1.2. Detalles de implementación: 27

5.2. Prototipo 2: Volcado de la información a la ontología 30

5.2.1 Análisis y diseño: 30
5.2.2 Detalles de implementación: 35
5.2. Prototipos 3 y 4: Reconocimiento de conceptos en textos y dar información..... 42
5.3.1. Análisis y diseño 42
5.3.2 Detalles de implementación 42

6. Verificación y evaluación 49
6.1. Verificación 49
6.1.1. Pruebas unitarias para el Crawler 49
6.1.2. Pruebas unitarias para la creación de la ontología 49
6.1.3. Pruebas unitarias para el detector e información 50
6.2. Análisis de los resultados 51
6.3. Evaluación 55

7. Conclusiones y trabajo futuro 58
7.1 Conclusiones sobre el trabajo 58
7.2 Conclusiones personales 58
7.3. Trabajo futuro 59

Bibliografía 81

Índice de Figuras

Figura 2.1 Arquitectura MVC 3
Figura 2.2 Ciclo de vida elegido 3
Figura 2.3 Diagrama EDT 8
Figura 2.4 Gantt General Inicial..... 12
Figura 2.5 Gantt Fase Inicial..... 13
Figura 2.6 Gantt Planificación y documentación..... 14
Figura 2.7 Gantt Análisis y diseño Inicial..... 15
Figura 2.8 Gantt Implementación Inicial..... 16
Figura 2.9 Gantt Verificación y Evaluación..... 17

Figura 4.1 Jerarquía de actores..... 23
Figura 4.2 Diagrama de casos de uso..... 23

Figura 5.1 Diagrama de clases Crawler..... 25
Figura 5.2 Listado de tratamientos patientslikeme.com..... 25
Figura 5.3 Enfermedades de un tratamiento..... 26
Figura 5.4 Modelo de dominio curetogether.com..... 26
Figura 5.5 Modelo de dominio patientslikeme.com..... 27
Figura 5.6 Ejemplo página de tratamiento plm..... 28
Figura 5.7 Ejemplo síntomas enfermedad plm..... 29
Figura 5.8 Diagrama de clases Ontología..... 30
Figura 5.9 Reajuste de escalas síntomas..... 31
Figura 5.10 Reajuste de escalas tratamientos 31
Figura 5.11 Modelo de dominio ontología..... 32
Figura 5.12 Ejemplo relación ontología..... 33
Figura 5.13 Object Properties ontología..... 33
Figura 5.14 Clases ontología..... 34
Figura 5.15 Diagrama de relaciones ontología..... 34
Figura 5.16 Diagrama de clases prototipos 3 y 4..... 42
Figura 5.17 Ejemplo de resultado..... 45
Figura 5.18 Ejemplo de resultado 2..... 47

Figura 6.1 Verificación del prototipo 1: Crawler..... 48
Figura 6.2 Verificación del prototipo 2: Ontología..... 48
Figura 6.3 Verificación del prototipo 3: Detector..... 49
Figura 6.4 Verificación del prototipo 4: Información..... 49
Figura 6.5 Ejemplo consulta SPARQL en Apache Jena Fuseki..... 51
Figura 6.6 Índice del etiquetado..... 52

Figura 8.1 Caso de uso extendido: Obtener información de la web 60
Figura 8.2 Interfaz gráfica: Obtener información de la web..... 61
Figura 8.3 Diagrama de secuencia: Obtener información de la web..... 61
Figura 8.4 Caso de uso extendido: Volcar la información a la Ontología..... 62
Figura 8.5 Interfaz gráfica: Volcar la información a la Ontología..... 62
Figura 8.6 Diagrama de secuencia: Volcar la información a la Ontología..... 63
Figura 8.7 Caso de uso extendido: Dar información de un texto..... 64

Figura 8.8 Caso de uso extendido: Dar información de un texto……. 64
Figura 8.9 Caso de uso extendido: Dar información de un texto (Detectar términos)……..…. 65
Figura 8.10 Caso de uso extendido: Dar información de un texto (Clasificar términos)……..…... 66
Figura 8.11 Caso de uso extendido: Dar información de un texto (Imprimir información)……. 67

Figura 9.1 Selección formato ontología……..…….…… 69
Figura 9.2 Añadir nueva clase Protégé…… 69
Figura 9.3 Eliminar una clase Protégé……. 70
Figura 9.4 Añadir Object Property Protégé……..... 71
Figura 9.5 Editar Object Property Protégé……..……..… 71
Figura 9.6 Eliminar Object Property Protégé……..…… 73
Figura 9.7 Añadir Data Property Protégé……..……. 73
Figura 9.8 Editar Data Property Protégé……..…….. 74
Figura 9.9 Añadir instancia Protégé……..…….. 75
Figura 9.10 Editar instancia Protégé……..……..…… 75
Figura 9.11 Editar instancia Protégé……..……..…… 76
Figura 9.12 Editar instancia Protégé……..……..…… 76
Figura 9.13 Editar instancia Protégé……..……..…… 77

Figura 10.1 Pantalla 3 del sistema……..…… 79
Figura 10.2 Pantalla de resultado del texto……..…... 80
Figura 10.3 Pantalla informativa de guardado……..……. 80

Índice de Tablas

Tabla 2.1 Modificaciones en diseños de página…… 18
Tabla 2.2 Las webs pasan a ser dinámicas…….... 19
Tabla 2.3 Bloqueo captcha……..... 19
Tabla 2.4 Problemas con versiones…….…… 19
Tabla 2.5 Problemas con el equipo……..…… 19
Tabla 2.6 Problemas por falta de conocimiento…… 19
Tabla 2.7 Problemas de salud…… 20
Tabla 2.8 Problemas ajenos al proyecto…… 20
Tabla 2.9 Mala planificación temporal……..…… 20

Tabla 5.1 Cambio de caracteres inválidos……..……..…… 36

Tabla 6.1 Evaluación del detector……..……..…… 56

Tabla 8.1 Caso de uso extendido: Obtener información de la web…….. 60
Tabla 8.2 Caso de uso extendido: Volcar la información a la Ontología…… 62
Tabla 8.3 Caso de uso extendido: Dar información de un texto……..…… 64

1. Introducción

El principal problema que aborda este proyecto es la falta de información que se obtienen de los test clínicos, ya que éstos son reducidos. Muchos de los síntomas o efectos adversos de un medicamento no han sido reportados durante los ensayos, y se identifican a través de los médicos de familia cuando ya se han manifestado.

Existen páginas web de contenido médico en las que los pacientes pueden compartir información acerca de sus enfermedades, causas, síntomas, tratamientos y efectos secundarios con otras personas. A partir de este tipo de páginas se puede obtener información adicional a los ensayos clínicos la cual se puede usar para descubrir nuevos síntomas o efectos de un tratamiento que no han sido reportados nunca en un test clínico. Las páginas a tratar serán curetogether.com y patientslikeme.com.

El objetivo de este proyecto consistirá en recopilar y procesar toda la información posible de estas páginas para crear una estructura de datos de la cual poder dar información a partir de textos que escribe la gente.

2. Planteamiento inicial

2.1. Objetivos

2.1.1 Objetivos principales

Este proyecto tiene dos objetivos principales a desarrollar:

El primero consistirá en crear una ontología en la que se recopilará y filtrará toda la información posible sobre enfermedades, tratamientos, causas… etc. de páginas webs. Par ello habrá que analizar las estructuras de estas webs médicas, adaptarlas a un único modelo de dominio a partir del cual crear la base de datos.

Por otra parte, se implementará una aplicación en la que un usuario podrá introducir un texto contando sus experiencias médicas, y a partir de este se le mostrará información que pueda ser de interés, consultando la base de datos ontológica que se ha creado

2.1.2 Objetivos personales

El objetivo principal de cualquier estudiante de grado que se enfrenta a este proyecto consiste en dar por finalizada la carrera. Además, tenía como objetivo personal realizar el trabajo de fin de grado en un lenguaje de programación diferente para poder llegar a dominar otro lenguaje, aparte de Java que es el que ha sido usado en gran parte durante toda la carrera. Por otra parte, el saber que el proyecto estaba relacionado con la medicina supuso una motivación extra, ya que la salud es lo más importante para cualquier persona, y el uso de sistemas de información en el entorno médico es algo que está creciendo exponencialmente hoy en día.

2.2. Arquitectura

Para la implementación de este proyecto, se va a hacer uso de una arquitectura mixta. Por un lado, usaremos una arquitectura local donde tanto la parte correspondiente al software como la relacionada con el almacenamiento de datos serán almacenadas en el equipo desde el que se vaya a implementar y ejecutar la aplicación. Por otro lado, será necesario es uso de una conexión a Internet para descargar los datos de las páginas web médicas.

Este tipo de arquitectura ha sido elegido para que se puedan descargar los datos de las webs ya que existe la posibilidad de que los datos sean borrados o las páginas web no estén accesibles. Una vez los datos hayan sido recolectados y guardados en el equipo, podrán ser procesados de forma local.

Por último, también se va a hacer uso de la arquitectura Modelo Vista Controlador (MVC), donde se crean tres estructuras independientes: una donde el sistema opera y gestiona los accesos a la información (Modelo), otra donde el usuario puede interactuar con la aplicación (Vista) y la última donde se gestiona las peticiones que se hacen al modelo y las respuestas que ofrece.

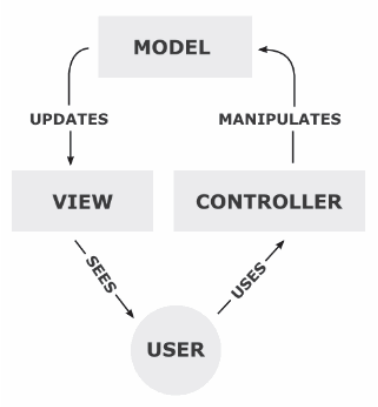


Figura 2.1 Arquitectura MVC

2.3. Alcance del proyecto

2.3.1. Ciclo de vida

El desarrollo del proyecto se realizará de forma iterativa e incremental, es decir, se dividirá el proyecto en módulos o prototipos que tengan una funcionalidad concreta y en cada iteración desarrollará uno de esos módulos y se añadirá al sistema. De esta manera se va obteniendo un sistema que va añadiendo incrementalmente las funcionalidades que necesita para su correcta ejecución.

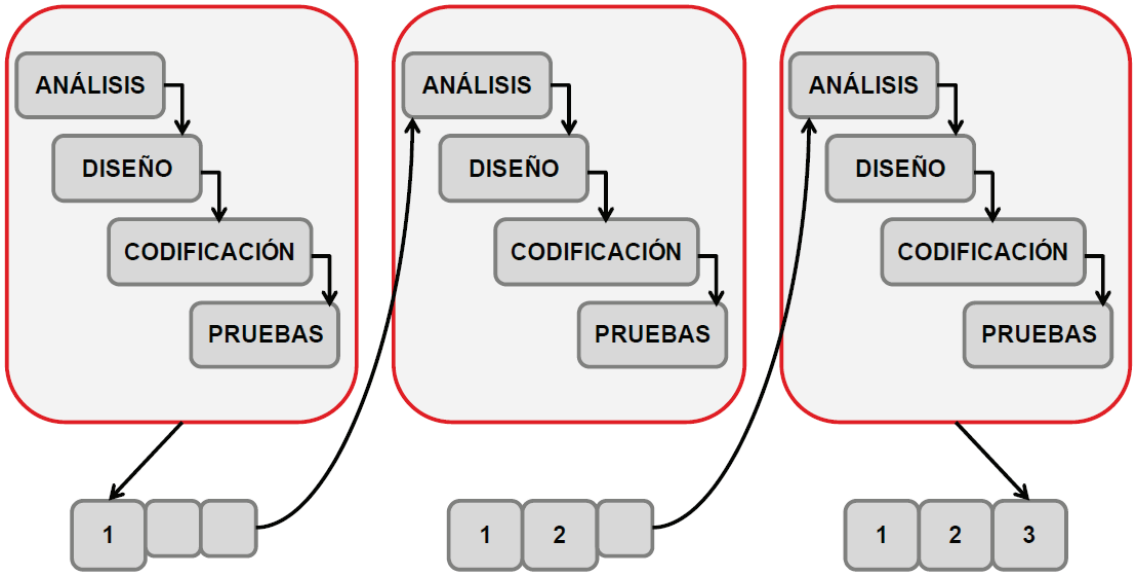


Figura 2.2 Ciclo de vida elegido

Se ha elegido ésta metodología de trabajo, porque al dividir un proyecto grande en prototipos

Se he elegido esta metodología porque tiene varias ventajas, que son las siguientes:

- Los errores solo afectarán al prototipo en el que se encuentre
- Es más fácil y tiene menos riesgos l hacer varios programas pequeños que uno grande
- Se controla mejor la calidad del producto al tener una nueva versión después de cada iteración

Como desventaja, mencionar que esta metodología requiere de una gestión más complicada.

2.3.2. Prototipos

A continuación, se van a describir los prototipos en los que se va a dividir la aplicación. Se va a dividir en cuatro prototipos, ya que como se ha descrito en los objetivos, hay cuatro grandes funcionalidades: recopilar información de las webs médicas, volcar la información en una ontología, detectar posibles términos en textos, y finalmente proporcionar información en base a esos términos. Por lo tanto, los prototipos a desarrollar son los siguientes:

- **Prototipo 1: Recopilación de información de webs**

  Este primer prototipo consistirá en analizar la estructura de la información de las webs médicas, y tratar de recopilar la mayor cantidad de información posible con una calidad mínima.

- **Prototipo 2: Creación de la ontología médica**

  En el segundo prototipo se diseñará y creará una ontología a partir de la información previamente obtenida. Para ello será necesario crear un modelo de dominio que se adecúe a la información de las webs y que se adaptará para crear la ontología.

- **Prototipo 3: Detector de términos**

  Una vez creada la ontología, se procederá a implementar un detector de términos médicos. Este detector se encargará de analizar textos que la gente haya compartido en estas webs y localizar términos médicos relevantes

- **Prototipo 4: Generación de información**

  Finalmente, el último prototipo consistirá en crear un programa que, a partir de los términos encontrados, muestre información sobre estos términos que sea de utilidad (efectividad de tratamientos, probabilidad de efectos secundarios… etc.)

2.4.Herramientas

Para el desarrollo de este proyecto se hará uso del lenguaje Python 3.6 para realizar los programas necesarios. Éstos se implementarán en el entorno de desarrollo JetBrains Pycharm.

Como estructura de datos para guardar la información médica, se usará una ontología. Para gestionar esta estructura de datos se usará la aplicación Protégé versión 5.0. Se ha elegido esta aplicación porque resulta bastante intuitivo y fácil de usar, y aparte es de código abierto.

Finalmente, se han usarán una serie de librerías externas para realizar los prototipos mencionados. Primero, en la recopilación de información de webs se hará uso de BeautifulSoup. A la hora de obtener información de la ontología se usará la librería SPARQLWrapper para realizar las consultas necesarias. Para usar esta librería, es necesario tener la ontolgía en un servidor. Esto se ha solucionado albergando la ontología en un servidor local usando la aplicación Apache Jena Fuseki.

En cuanto a la documentació, se ha hecho uso de Visual Paradigm "Comunity version" para crear los diagramas de clase, secuencia y casos de uso. Esta versión es totalmente gratuita, y eso por esto por lo que se ha elegido esta herramienta. En cuanto a la redacción de la memoria se ha usado Microsoft Word 2016 ya que venía instalado en el ordenador, y por último se ha usado Gantt Project para elaborar el diagramana de Gantt.

- **Protégé**.

Protégé en una herramienta de código abierto para la creación y edición de ontologías (Cal Poly , 2011). A continuación, se muestra las diferentes secciones que tiene, y que son más relevantes para el proyecto:

- **Classes*:*** En esta en esta sección se crean, y editan las clases de la ontología. Se puede especificar las propiedades de cada clase (*subclass Of, Equivalent To...*), añadir subclases, relaciones...

- **Object properties*:*** Representa las relaciones entre clases. Todas las diferentes secciones de Protégé tienen una estructura similar. Se crean las relaciones de la misma manera que en la sección de clases. Se puede especificar el Dominio y el Rango de cada relación, indicando de qué tipo tienen que ser las clases que participen en la relación. Hay que tener en cuenta que, si se especifica un dominio de una clase en concreto, al indicar que una instancia es de ese dominio, Protégé inferirá automáticamente que esa instancia es de la clase del dominio. Esto puede dar problemas de consistencia si no se tiene en cuenta.

  En la parte *Characteristics* se puede especificar si la relación cumple alguna propiedad de las siguientes:

  - Functional: la relación sólo puede tener un miembro en el rango por cada miembro del dominio. (Ej.: *Person* hasGender *Gender* sería funcional, ya que cada persona(dominio) sólo puede tener un género(rango))
  - Inverse functional: la relación sólo puede tener un miembro en el dominio por cada miembro del rango. (Ej.: *Gender* isGenderOf *Person* sería inversamente funcional)
  - Transitive: si "a aProperty b" y "b aProperty c" entonces "a aProperty c".
  - Symmetric: si "x aProperty b" entonces se cumple también "b aProperty a"
  - Asymmetric: si "x aProperty b" entonces "b aProperty a" no sería cierto.

- Reflexive: Significa que “x aProperty x” es siempre cierto.
- Irreflexive: Signicica que “x aProperty x” nunca puede ser cierto.

- **Individuals** Son las instancias de la ontología. En la parte Description se indica de qué clases son cada instancia. Por ejemplo, Depression sería de la clase enfermedad y Sadness de la clase síntoma.
  En la parte Property assertions se especifican las Object y Data properties que tiene. Por ejemplo, en la instancia Depression, habría que añadir Object propertie tiene_sintoma Sadness.

- **Data properties** Representan atributos asociados a una clase. Cada instancia de la clase a la que pertenezca dicho data propertie puede tener un valor concreto diferente para aportar información extra a la ontología.

➢ JetBrains Pycharm

JetBrains Pycharm (https://www.jetbrains.com/pycharm/) es un entorno de desarrollo integrado (IDE) utilizado para la programación en lenguaje Python. Se ha elegido esta plataforma porque se ha usado previamente durante la carrera y porque usando un usuario de estudiante de universidad, se puede obtener una licencia de cualquier tipo de manera gratuita. (JetBrains, 2017)

➢ BeautifulSoup

BeautifulSoup (https://pypi.python.org/pypi/beautifulsoup4) es una biblioteca para Python usad para la extracción de datos de archivos HTML y XML. Ésta crea un árbol con todos los elementos del documento de forma que facilita la extracción de información de éste, muy útil para realizar tareas de web scraping. (Python Software Foundation, 2017)

➢ Apache Jena Fuseki

Apache Jena Fuseki es un servidor para SPARQL. Puede funcionar como un servicio de sistema operativo, como una aplicación web Java (archivo WAR), y como un servidor independiente. Se proporciona seguridad (usando Apache Shiro) y tiene una interfaz de usuario para la supervisión y administración del servidor.

Proporciona el protocolos para la consulta y actualización SPARQL 1.1 , así como el protocolo SPARQL Graph Store . Se puede utilizar para proporcionar el motor de protocolo para otros sistemas de consulta y almacenamiento RDF. (Apache Sowftware Foundation, 2017).

➢ SPARQLWrapper.

SPARQLWrapper es una biblioteca de Python que permite poder realizar consultas SPARQL contra un servidor y convierte el resultado en un formato manejable para gestionar los resultados obtenidos. (Python Software Foundation, 2014)

2.1 Planificación temporal

2.4.1. Etapas

En este apartado, se describen las tareas en las que se ha dividido el trabajo. El proyecto se dividirá en cuatro prototipos, tal y como se ha descrito anteriormente. Cada tendrá una estructura de desarrollo similar. A continuación, se describen las tareas principales, y posteriormente se mostrará el diagrama EDT

- **Planificación Inicial**

  La primera tarea del proyecto consiste en exponer la idea del proyecto y definir los posibles objetivos y alcance del proyecto. También se decidirá que posibles estructuras se pueden usar para gestionar la información, así como el lenguaje de programación en el que se desarrollará la aplicación.

  Posteriormente se realizará una planificación temporal del proyecto, como estudio de antecedentes, EDT, Gantt…. etc.

- **Aprendizaje**

  Durante esta etapa se realiza la formación necesaria sobre las herramientas y tecnologías necesarias para el desarrollo de cada prototipo del proyecto.

- **Análisis y diseño**

  Durante esta etapa, primero se buscan posibles soluciones para la funcionalidad de cada prototipo y posteriormente se realizará el diseño necesario para la correcta implementación de cada uno de estos.

- **Implementación**

  En esta etapa se implementará cada prototipo, previamente diseñado en la etapa anterior

- **Documentación**

  Durante la realización de cada prototipo del proyecto se irá generando la documentación necesaria para la realización de la memoria del proyecto.

- **Presentación**

  Una vez finalizado el proyecto, se realizará la preparación de la presentación.

2.4.2. Estructura de Desglose de Trabajo (EDT)

En la siguiente imagen se puede observar el diagrama EDT que representa la estructura de trabajo que se ha llevado a cabo durante el desarrollo del trabajo

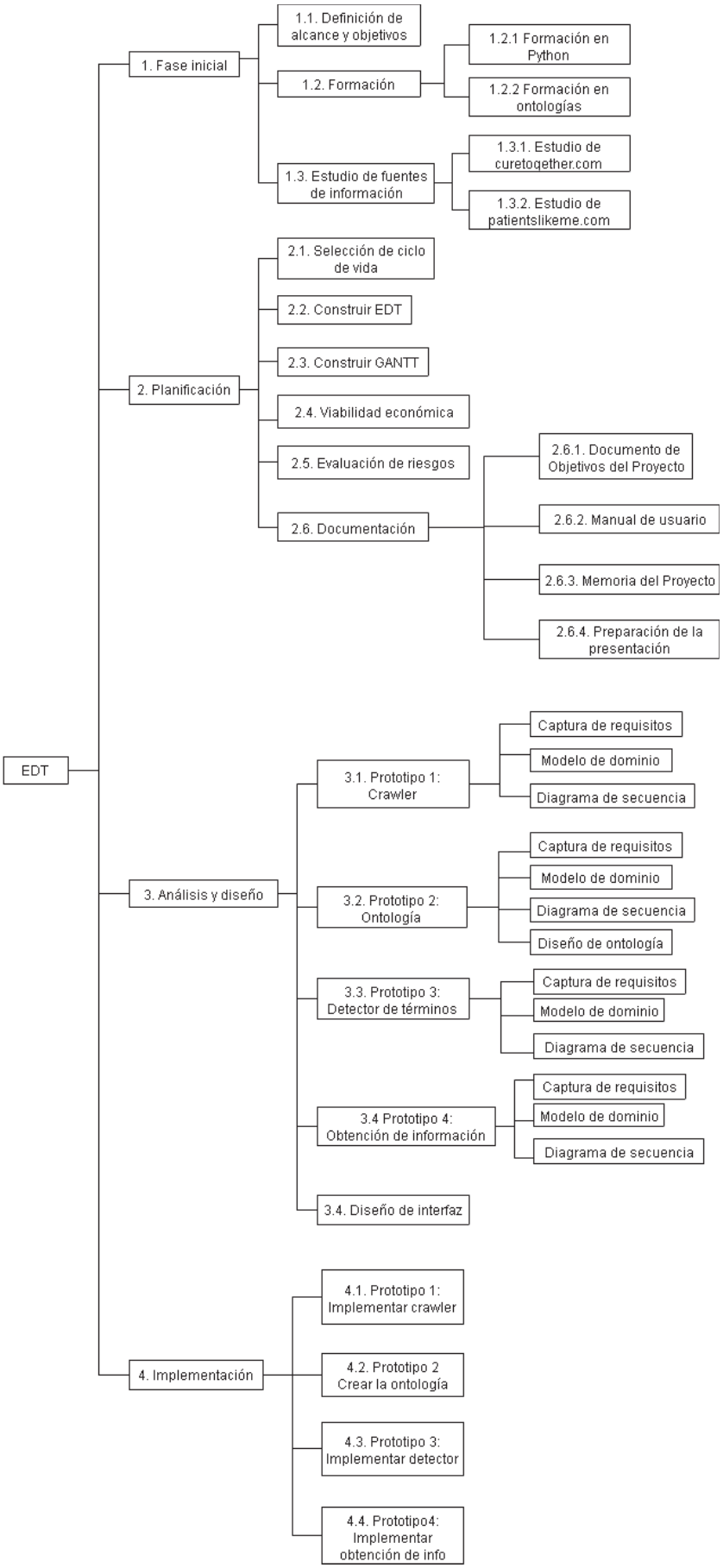


Figura 2.3 Diagrama EDT

2.4.3. Descripción de tareas

Tarea 1. Fase inicial

Tarea 1.1. Definir alcance y objetivos del proyecto

En esta etapa de definen los objetivos del proyecto y cuál va a ser su alcance

Tarea 1.2. Formación

Ya que se trataba de un lenguaje de programación y sistema de base de datos nuevo, ha sido necesario un periodo de formación en el que se ha aprendido a programar en Python, y que es una ontología y cómo se puede adaptar el uso de ésta al proyecto.

Tarea 1.3. Estudio de fuentes de información

Durante esta tarea se han estudiado las dos webs médicas que se van a tratar para saber qué y cuánta información se puede obtener de ellas.

Tarea 2. Planificación

Esta etapa tiene lugar durante todo el desarrollo del trabajo, ya que abarca tanto la planificación, redacción del documento de objetivos del proyecto, redacción de la memoria, y las conclusiones finales.

Tarea 2.1. Elección de ciclo de vida

Se define el ciclo de vida que va a seguir el proyecto

Tarea 2.2. Construir EDT

En esta tarea se desglosan todas las actividades de las que consta el proyecto, y posteriormente se realizará una descripción de cada una.

Tarea 2.3. Construir Gantt

Tras construir el EDT, se realizará una estimación de temporal de cada una de las tareas definidas, y se generará un diagrama de GANTT

Tarea 2.3.1. Construir Gantt inicial

Se estima los tiempos de las tareas antes de comenzar el desarrollo del proyecto para tener una estimación temporal de la duración de éste.

Tarea 2.3.2. Construir Gantt final

Una vez finalizado el proyecto, se realizará otro diagrama con la duración real de cada actividad

Tarea 2.5. Evaluación de riesgos

En esta sección se definen los posibles riegos que pueden surgir durante el desarrollo del trabajo, y se define un plan de prevención y otro de contingencia para cada uno de ellos y así minimizar su impacto.

Tarea 2.6. Documentación

Esta tarea consiste en documentar todo el trabajo realizado, es decir, la idea principal, qué decisiones se han tomado durante el desarrollo, que problemas han surgido, que conclusiones se han obtenido, etc.

Tarea 2.6.1. Documento de objetivos del proyecto

Consiste en generar un documento que describe los objetivos del proyecto, y como se van a desarrollar.

Tarea 2.6.2. Manual de usuario

Describe las funcionalidades de la aplicación, y como usarla correctamente

Tarea 2.6.3. Memoria

En este proceso se describe el proceso de desarrollo del proyecto, se exponen las conclusiones y se da por finalizado

Tarea 2.6.4. Preparación de la defensa

En este periodo se realizará la exposición que se va a presentar y se preparará la defensa del proyecto

Tarea 3. Análisis y diseño

Para cada prototipo habrá que realizar las tareas de análisis y diseño que se requieran

Tarea 3.1. Prototipo 1: Crawler

Este primer prototipo consistirá en desarrollar un "web crawler" que extraerá toda la información posible de las páginas médicas en ficheros de texto. Por lo que será necesario hacer los modelos de domino de cada página web.

Tarea 3.2. Prototipo 2: Crear Ontología

Una vez se haya recopilado toda la información posible se procederá a crear la ontología. Para ello es necesario crear una única estructura de datos, por lo que habrá que analizar las diferencias entre los modelos de domino de las webs, y adaptarlas a un único modelo de domino que será el que tendrá la ontología.

Tarea 3.3. Prototipo 3: Detector

Este prototipo se encargará de, dado un texto, detectar posibles términos que se encuentren en la ontología, y clasificarlos por tipo(enfermedad, síntoma ,tratamiento, etc.). Para ello será necesario poder acceder a los datos de la ontología para compararlos con los del texto y encontrar posibles términos

Tarea 3.4 Prototipo 4: Obtención de información

Este último prototipo consistirá en generar información a partir de los datos que nos proporcione el detector y con la información albergada en la ontología.

Tarea 3.4. Diseño de interfaz

Se diseñará una interfaz de usuario en la que se podrán ejecutar todas las funcionalidades del proyecto

Tarea 4. Implementación

Tarea 4.1. Implementar prototipo 1: Crawler

En esta primera fase de implementación, se desarrollará el crawler para la recogida de información.

Tarea 4.2. Implementar prototipo 2: Crear Ontología

En la segunda tarea de implementación, se creará la ontología con los datos obtenidos del crawler. Para ello habrá que extraer la información que sea necesaria de los ficheros resultado y volcarlos en la ontología

Tarea 4.3. Implementar prototipo 3: Detector

En esta tarea, una vez creada la ontología, se implementará el detector de términos.

Tarea 4.4. Implementar prototipo 4: Obtención de información

En esta última tarea se implementará el algoritmo que genere la información a partir de los términos encontrados

2.4.4. Diagrama de Gantt Inicial

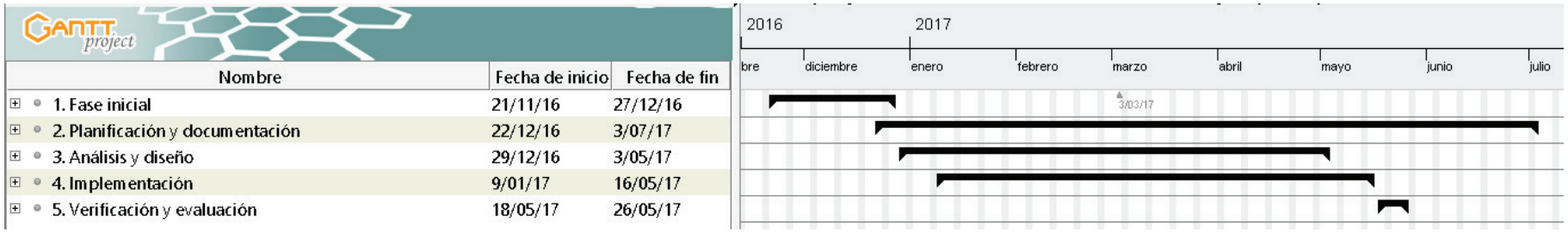


Figura 2.4: Gantt General

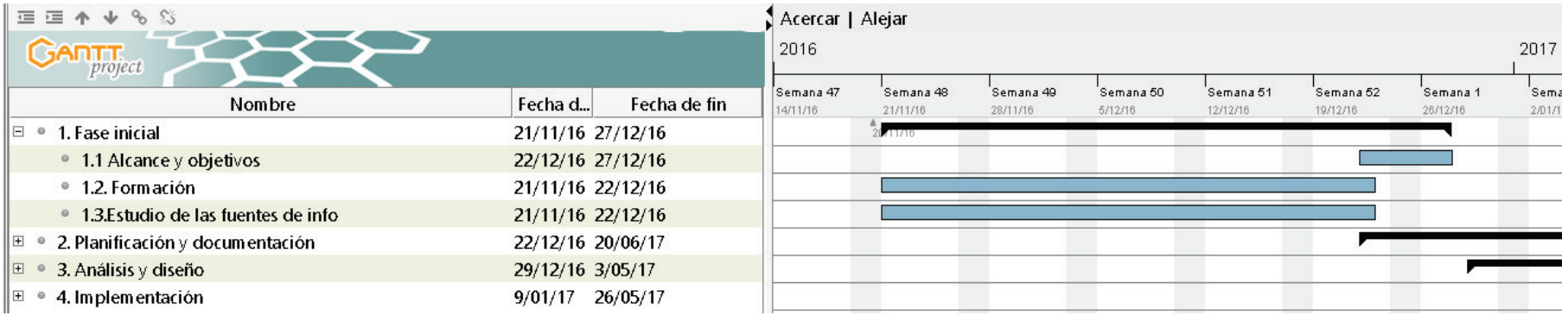
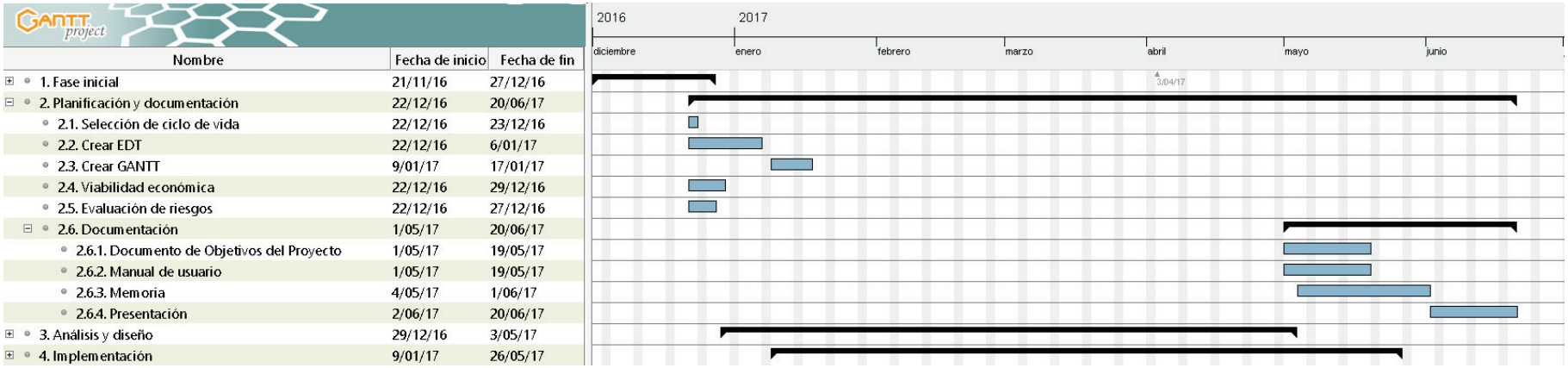


Figura 2.5: Gantt Fase Inicial



Nombre	Fecha de inicio	Fecha de fin
1. Fase inicial	21/11/16	27/12/16
2. Planificación y documentación	22/12/16	20/06/17
2.1. Selección de ciclo de vida	22/12/16	23/12/16
2.2. Crear EDT	22/12/16	6/01/17
2.3. Crear GANTT	9/01/17	17/01/17
2.4. Viabilidad económica	22/12/16	29/12/16
2.5. Evaluación de riesgos	22/12/16	27/12/16
2.6. Documentación	1/05/17	20/06/17
2.6.1. Documento de Objetivos del Proyecto	1/05/17	19/05/17
2.6.2. Manual de usuario	1/05/17	19/05/17
2.6.3. Memoria	4/05/17	1/06/17
2.6.4. Presentación	2/06/17	20/06/17
3. Análisis y diseño	29/12/16	3/05/17
4. Implementación	9/01/17	26/05/17

Figura 2.6: Gantt Planificación y documentación Inicial

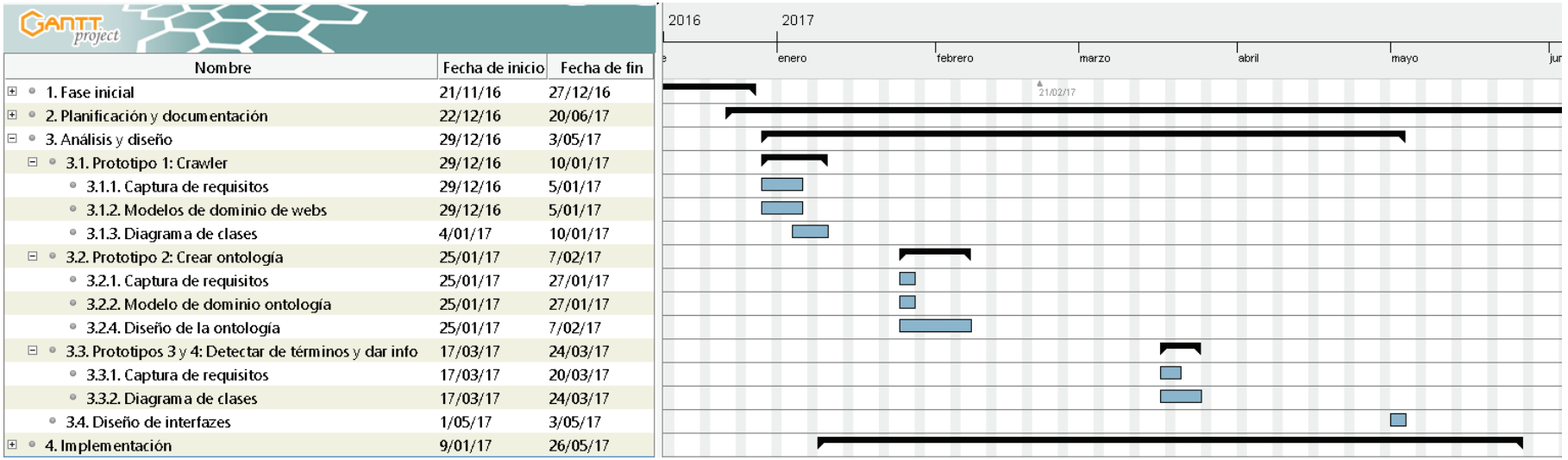


Figura 2.7: Gantt Análisis y diseño inicial

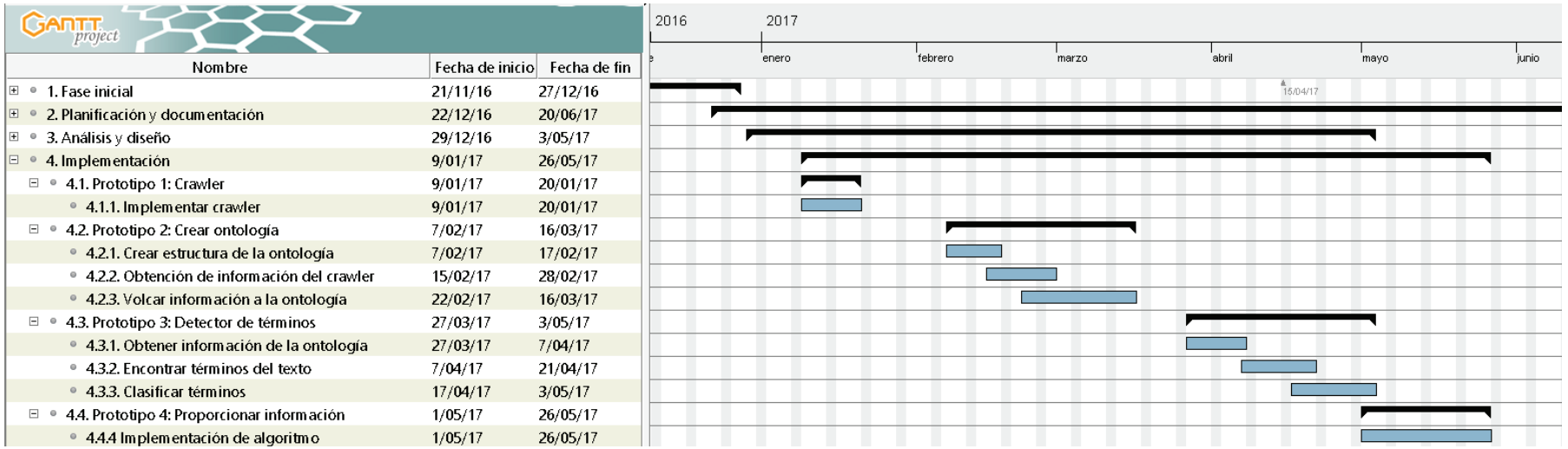


Figura 2.8: Gantt Implementación Inicial

Figura 2.9: Gantt Verificación y Evaluación

2.4.5. Diagrama de Gantt Final

Durante el desarrollo del proyecto se han ido cumpliendo los tiempos establecidos casi por completo, por lo que el diagrama de Gantt final es equivalente al inicial.

2.1 Gestión de riesgos

En este punto se van a identificar los posibles riesgos que pueden afectar al correcto desarrollo del proyecto. A demás se definirá un plan de prevención y de contingencia para intentar paliar los efectos de los posibles inconvenientes que surjan. A continuación, se enumeran los riesgos que se han tenido en cuenta:

- **Modificaciones en el diseño de páginas.** Si durante el proceso de obtención de información de las webs, estas cambian su diseño, y por tanto, su código HTML, habrá que volver a implementar el crawler.
- **Las webs pasan a ser dinámicas.** Si durante el proceso de obtención de información de las webs, pasan a ser dinámicas y usar javascripts, habría que estudiar cómo se podría obtener de nuevo esta información.
- **Bloqueo captcha.** Si durante el proceso de obtención de información de las webs, están introducen un bloqueo de seguridad como el captcha, no será posible acceder a su contenido por una máquina, por lo que no se podría obtener la información.
- **Problemas de versiones.** Puede ocurrir que distintas versiones, tanto de Python como de sus librerías o Protégé, tengas algunos problemas en las versiones, o no sean compatibles unas con otras.
- **Problemas con Protégé.** Ya que se trata de una herramienta que no se conoce, pueden surgir problemas como que se tarda más de lo previsto en aprender a usarla o que hay ciertas funcionalidades que no cumple.
- **Problemas con el equipo.** Puede suceder que, durante el desarrollo del proyecto, un virus afecte al equipo o que se averíe por golpes o causas similares, lo que supondría la pérdida de información.
- **Problemas por falta de conocimiento**. Ya que se trata de un proyecto donde se usan lenguajes y estructuras de datos que no se han visto durante la carrera, la falta de conocimiento puede hacer que el número de horas se incremente.
- **Problemas de salud.** Durante la elaboración del trabajo, pueden surgir enfermedades que, dependiendo de la gravedad, afectarán más o menos a la finalización del proyecto.
- **Problemas ajenos al proyecto.** Pueden surgir compromisos durante el proyecto, como imprevistos familiares, viajes no planeados, entrevistas de trabajo… que pueden retrasar la entrega del trabajo.
- **Problemas de planificación.** Realizar una mala planificación puede suponer la entrega fuera de plazos del proyecto.

Las siguientes tablas muestran para cada riesgo, la probabilidad y efecto, junto con su plan de contingencia y prevención.

Riesgo	**Modificaciones en diseños de página**
Efecto	Retraso en la entrega del proyecto
Impacto	Muy alto
Probabilidad	Baja
Plan de prevención	No se puede prevenir
Plan de contingencia	Volver a implementar el crawler teniendo en cuenta el nuevo diseño

Tabla 2.1: Modificaciones en diseños de página

Riesgo	Las webs pasan a ser dinámicas.
Efecto	Retraso en la entrega del proyecto
Impacto	Muy alto
Probabilidad	Muy baja
Plan de prevención	No se puede prevenir
Plan de contingencia	Estudiar si se puede obtener la información con otras técnicas, y si no, buscar otras páginas médicas.

Tabla 2.2: Las webs pasan a ser dinámicas

Riesgo	Bloqueo captcha
Efecto	Retraso en la entrega del proyecto
Impacto	Muy alto
Probabilidad	Muy baja
Plan de prevención	No se puede prevenir
Plan de contingencia	Buscar otras páginas médicas accesibles.

Tabla 2.3: Bloqueo captcha

Riesgo	Problemas con versiones
Efecto	Retraso en la entrega del proyecto
Impacto	Medio
Probabilidad	Alta
Plan de prevención	Comprobar que las versiones que se van a utilizar son completamente funcionales.
Plan de contingencia	Cambiar a una versión más estable

Tabla 2.4: Problemas con versiones

Riesgo	Problemas con el equipo
Efecto	Pérdida de información
Impacto	Muy alto
Probabilidad	Media
Plan de prevención	Realizar copias de seguridad para evitar que la pérdida de información si se pierde un equipo. Tener un buen antivirus correctamente actualizado y ser cuidadoso con el equipo
Plan de contingencia	Adquirir un nuevo equipo o usar otro si es posible y añadir las copias de seguridad del proyecto.

Tabla 2.5: Problemas con el equipo

Riesgo	Problemas por falta de conocimiento
Efecto	Incremento del número de horas necesarias para cumplir con los plazos de entrega
Impacto	Alto
Probabilidad	Muy alta
Plan de prevención	Conseguir buena documentación complementaria
Plan de contingencia	Dedicar horas extras al aprendizaje necesario sobre el tema

Tabla 2.6: Problemas por falta de conocimiento

Riesgo	Problemas de salud
Efecto	Retraso en la entrega del proyecto
Impacto	Alto
Probabilidad	Alta
Plan de prevención	Cuidarse adecuadamente.
Plan de contingencia	Dedicar horas extra una vez superada la enfermedad.

Tabla 2.7: Problemas de salud

Riesgo	Problemas ajenos al proyecto
Efecto	Retraso en la entrega del proyecto
Impacto	Alto
Probabilidad	Alta
Plan de prevención	Este tipo de improvistos no es posible prevenirlos
Plan de contingencia	Dedicar horas extra al proyecto.

Tabla 2.8: Problemas ajenos al proyecto

Riesgo	Mala planificación temporal
Efecto	Retraso en la entrega del proyecto
Impacto	Medio
Probabilidad	Alta
Plan de prevención	Realizar una buena planificación.
Plan de contingencia	Modificar la planificación temporal y dedicarle más horas al proyecto

Tabla 2.9: Mala planificación temporal

3. Estado del arte

3.1 Ontología

El sistema de datos que se va a usar para guardar la información del crawler una vez procesada será una ontología. En el sentido filosófico de la palabra, la ontología es el estudio del conocimiento y la existencia tal y como es percibido por los humanos.

Para explicar qué es una ontología en el ámbito computacional existen dos definiciones posibles.

"Una especificación formal y explícita de una conceptualización compartida**"** (Guber 1993). Dicho de otra forma, es un conjunto de conceptos y relaciones definidos explícitamente, legible para una máquina, que representa el conocimiento de un dominio y facilita la compartición de información entre sistemas. Esta descripción está orienta a inteligencia artificial.

La segunda, más orientada a SI, dice así: "Es un artefacto del software diseñado para un conjunto específico de usos y ambientes computacionales" (Guarino 2005). En el caso de este proyecto, el *conjunto específico de usos* consistirá en guardar la información obtenida y procesada de las webs médicas para posteriormente consultar en ella.

Componentes de una ontología

- Clases **:** ideas básicas que se intentan formalizar.
- Relaciones: representan la interacción entre clases. Forma la taxonomía del dominio, una relación tiene un dominio y un rango(los nodos de la relación)
- Funciones**:** tipo concreto de relación, donde un elemento es el resultado de una función que considera varios elementos de la ontología.
- Instancias: representan objetos de un concepto.
- Axiomas**:** teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología. "Si X e Y son de la clase Z, entonces X no es subclase de Y".

En resumen, las ontologías se usan para especificar y comunicar el conocimiento del dominio de una manera genérica y son muy útiles para estructurar y definir el significado de los términos. El uso de ontologías en el desarrollo de los SI permite establecer correspondencia y relaciones entre los diferentes dominios de entidades de información.

(Graciela Barchini, Margarita Álvarez, Susana Herrera y Melina Trejo, 2007)

3.1 OWL

OWL (Web Ontology Languaje) es un modelo de datos de la familia de especificaciones de la World Wide Web (W3C) que ha sido diseñado para cubrir la necesidad de un lenguaje de ontologías Web (World Wide Web Consortium, 2009). Una web semántica se basa en la capacidad de XML para definir esquemas mediante etiquetas y en RDF para representar datos de forma flexible. El siguiente nivel por encima de esto es un lenguaje de ontologías que sea capaz de describir el significado de la terminología usada. OWL ha sido creado para las ontologías Web, forma parte de un conjunto de recomendaciones del W3C relacionadas con la web semántica.

- **XML** proporciona una sintaxis para estructurar documentos, pero no impone restricciones semánticas en el significado de estos documentos
- **XML Schema** se usa para restringir la estructura de los documentos XML y amplia los tipos de datos de XML

- **RDF** es una estructura de datos para objetos y relaciones entre éstos. También se puede definir como un formato de datos para grafos dirigidos y etiquetados para representar la información en la Web. Este tipo de lenguaje se puede representar también en una sintaxis XML
- **RDF Schema** se usa para describir propiedades y clases de recursos RDF, con una semántica para la generalización y jerarquización tanto de propiedades como de clases
- **OWL** añade vocabulario extra para describir propiedades y clases (relaciones, cardinalidad, características de propiedades, clases enumeradas, igualdad… etc)

OWL proporciona tres tipos de lenguajes, cada uno con un nivel de expresividad mayor. A continuación, se describen cada uno de ellos

- **OWL Lite** es el más simple. Diseñado para usuarios que necesitan una clasificación jerárquica y restricciones simples, por ejemplo, sólo permite relaciones de cardinalidad de 0, 1.
- **OWL DL** permite más expresividad garantizando que sea computable, y que los cálculos se realizarán en tiempo finito. Contiene todas las construcciones del lenguaje OWL pero tienen ciertas restricciones de uso (por ejemplo, una instancia no puede ser de dos clases a la vez, o una clase no puede ser una instancia de otra clase.)
- **OWL Full** permite máxima expresividad y libertad sintáctica de RDF pero no se garantiza que sea computable. Es poco probable que, por el momento, algún software de razonamiento sea capaz de obtener un razonamiento completo para cada característica de OWL Full.

En resumen, OWL Full se puede considerar como una extensión de RDF, mientras que los otros dos puede ser considerados como extensiones de una visión restringida de RDF. Cada documento OWL es un documento RDF y cada documento RDF es un documento OWL Full, pero solo algunos RDF serán legalmente OWL DL o Lite

3.1 SPARQL

SPARQL (Protocol and RDF Query Lenguaje) se define como un protocolo que permite expresar consultas en fuentes de datos en forma RDF. RDF (Resource Description Framework) se puede definir como un modelo de datos de la familia de especificaciones de la W3C (Worl Wide Web Consorcium, 2008). Este lenguaje se basa en hacer declaraciones sobre los recursos, en forma de expresiones sujeto-predicado-objeto, más conocidas como triples. El sujeto hace referencia al recurso en cuestión, el predicado especifica la propiedad o la relación, y finalmente el objeto es el valor que se le desea dar a al recurso, o el otro recurso con el que se desea establecer una relación.

La mayoría de consultas en SPARQL (Protocol and RDF Query Lenguaje) se realizan usando un conjunto de patrones de tripleta (triple patterns), llamados también patrón de grafo básico. Estos patrones son similares a las tripletas del lenguaje RDF (sujeto, objeto y predicado), solo que cada elemento puede ser una variable (?x). Al realizar una consulta simple, si los elementos de las variables coinciden con un subgrafo de un grado RDF, estos elementos son devueltos como resultado. A continuación se muestra un ejemplo de cómo sería la estructura de una consulta con una tripleta:

```
SELECT ?subject ?predicate ?object
            WHERE {
   ?subject ?predicate ?object
                }
```

4. Captura de requisitos

4.1 Jerarquía de actores

Para este proyecto únicamente habrá un tipo de actor, que será el usuario que interactúe con la vista de la aplicación, por lo que en la jerarquía de actores únicamente aparece el actor 'Usuario', como se muestra en la Figura 4.1:

Figura 4.1 Jerarquía de actores

4.1 Diagrama de casos de uso

En la siguiente figura (Figura 4.2) se muestran las funcionalidades de las que consta este proyecto:

1)

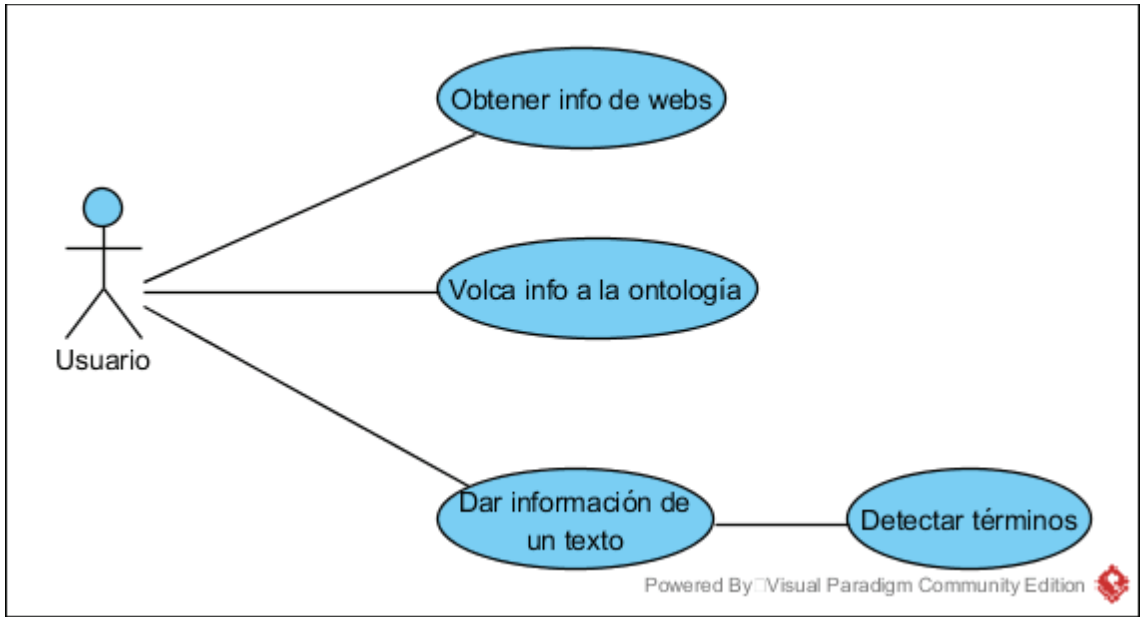


Figura 4.2: Diagrama de casos de uso

5. Desarrollo de prototipos

5.1 Prototipo 1: Captura y preproceso de información de internet

5.1.1 Análisis y diseño:

En este primer prototipo se encargará de extraer datos las webs de contenido médico. Para este primer prototipo se analiza en contenido de cada página para comprobar cuanta información se puede recoger. La información que se puede reportar en las páginas es la siguiente es la siguiente:

- Síntomas de una enfermedad (y un grado de severidad)
- Causas de una enfermedad (y un grado de severidad). Solo en curetogether.
- Tratamientos de una enfermedad (y un grado de mejora)
- Efectos secundarios de un tratamiento (y un grado de severidad)

De curetogether ya se disponía la información en ficheros .csv al inicio del proyecto de enfermedades, síntomas, tratamientos, efectos secundarios y causas. En concreto, los 4 ficheros que se describen debajo:

- **symptoms_data.csv**: cada línea contiene información de los grados de severidad de un síntoma para una enfermedad. Se muestra la siguiente línea de ejemplo: *1 people reported mild Burning nose,1 people reported moderate Burning nose,3 people reported severe Burning nose,2 people reported having Burning nose but did not provide a severity rating,69 people reported not having Burning nose,Burning nose,http://curetogether.com/acid-reflux/symptoms/;;;;;;;;*
  Cada grado de severidad se divide por comas, y finalmente en los dos últimos elementos de la línea se indica que síntoma es y la url de la enfermedad.

- **treatments_data.csv:** cada línea contiene información de los grados de efectividad de un tratamiento para una enfermedad. Se muestra la siguiente línea de ejemplo: *10 people thought Zinc cream had no effect on their Acne,8 people experienced a moderate improvement after trying Zinc cream,Zinc cream,http://curetogether.com/acne/treatments/*
  Cada grado de severidad se divide por comas, y los dos últimos elementos de la línea se corresponden con la enfermedad para la que se ha tratado y la url del tratamiento.

- **causes_data.csv:** cada línea contiene información la probabilidad de que esa causa haya provocado una enfermedad. Se muestra la siguiente línea de ejemplo: *241 out of 528 respondents (46%) think Being overweight is a cause of their Acid Reflux (GERD),Being overweight,http://curetogether.com/acid-reflux/causes/*
  El primer elemento corresponde al grado de probabilidad o de reportes que piensan que esa causa a provocado la enfermedad, el segundo elemento es la causa, y el último la url de la enfermedad.

- **side_effects_data.csv:** cada línea contiene información sobre los efectos secundarios de un tratamiento. Se muestra la siguiente línea de ejemplo: *31 out of 108 respondents (29%) experienced diarrhea with Doxycycline,15 out of 108 respondents (14%) experienced itching of the rectum or vagina with Doxycycline,9 out of 107 respondents (8%) experienced sore mouth with Doxycycline,4 out of 14*

respondents (29%) experienced Low Fever with Doxycycline,1 out of 17 respondents (6%) experienced cough with Doxycycline,Doxycycline,http://curetogether.com/acne/side-effects/

Los primeros elementos corresponden con el número de reportes de cada efecto secundario de tratamiento. Los dos últimos corresponden al tratamiento y a la url de la enfermedad de donde se ha obtenido la información. Realmente esta url no nos interesa, porque no hay relación entre la enfermedad y el efecto secundario del tratamiento, únicamente entre el tratamiento y el efecto secundario.

El crawler que se implementará debe recoger únicamente la información de patientslikeme.com

Como diseño, únicamente se necesita una clase para implementar el Crawler:

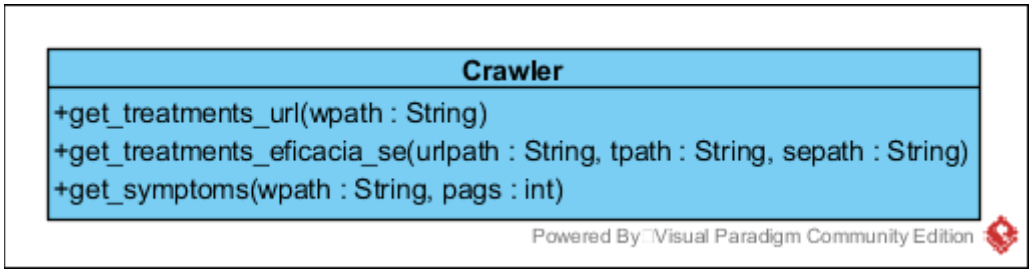


Figura 5.1: Diagrama de clases Crawler

El método *get_treatment_eficacia_se* recoge la información tanto de efectos secundarios de tratamientos como de efectividad contra enfermedades.

De patientslikeme no se ha extraído datos de causas, ya que no proporciona esta información.

En cuanto a los tratamientos, se ha decido extraer solamente información sobre los tratamientos que tuvieran más de diez casos o reportes. La web de patientslikeme ofrece un listado de todos los tratamientos, ordenados por número de evaluaciones (tal y como se muestra en la figura), lo que ha facilitado la tarea de extracción simplemente indicando hasta qué página del listado se deseaba extraer tratamientos.

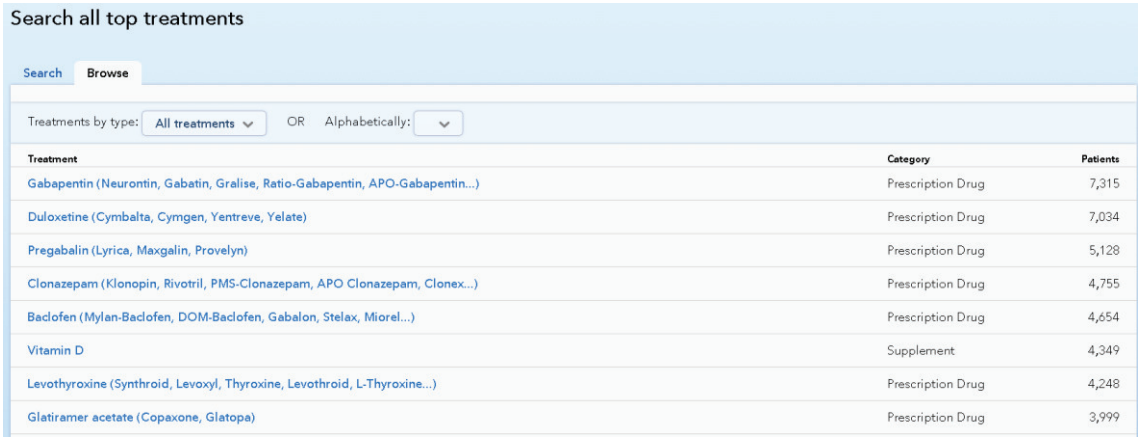


Figura 5.2: Listado tratamientos patientslikeme.com

Además, de cada tratamiento no se ha podido sacar todo el contenido ya que, cuando se recoge el contenido HTML de una página de un tratamiento (ej.https://www.patientslikeme.com/treatments/show/279), solo se obtiene información de los primeros "purposes" que hay en la lista de dicha página, tal y como se observa en la figura 5.3, luego hay un desplegable que si se pulsa, muestra más, pero esos no aparecen en el contenido

HTML que se obtiene. Lo bueno es que están ordenados por cantidad de evaluaciones, por lo que de cada uno se recoge los que más evaluaciones tienen.

Figura 5.3: "Purposes" de un tratamiento patientslikeme.com

Una vez analizada la información, lo primero de todo es construir los modelos de dominio de ambas páginas, para observar las similitudes y las diferencias entre éstos y finalmente poder adaptar la información a un único modelo de dominio, que será el que tenga la ontología.

A continuación, se muestran ambos modelos de domino, primero curetogether y después patientslikeme:

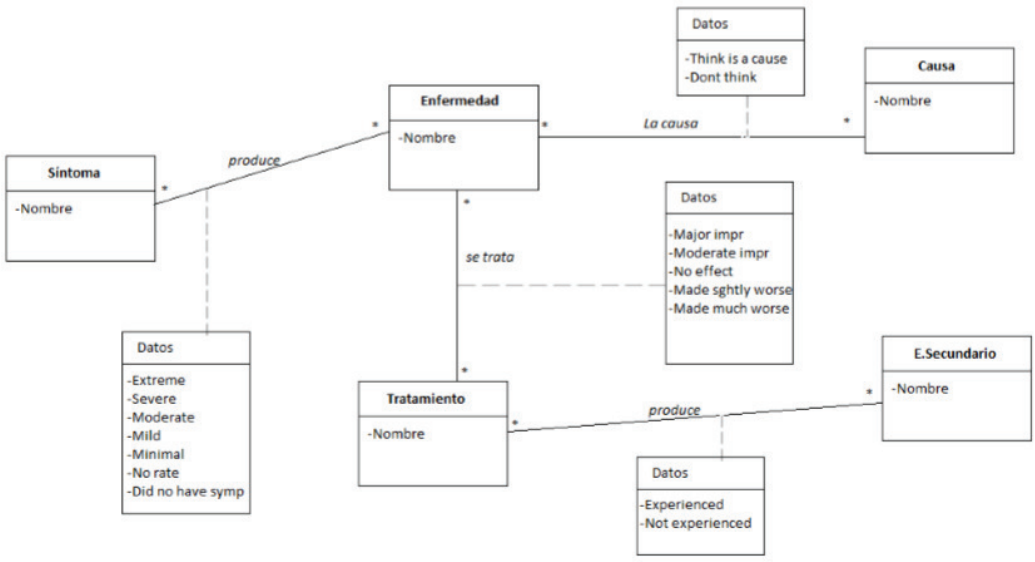


Figura 5.4: Modelo de dominio curetogether.com

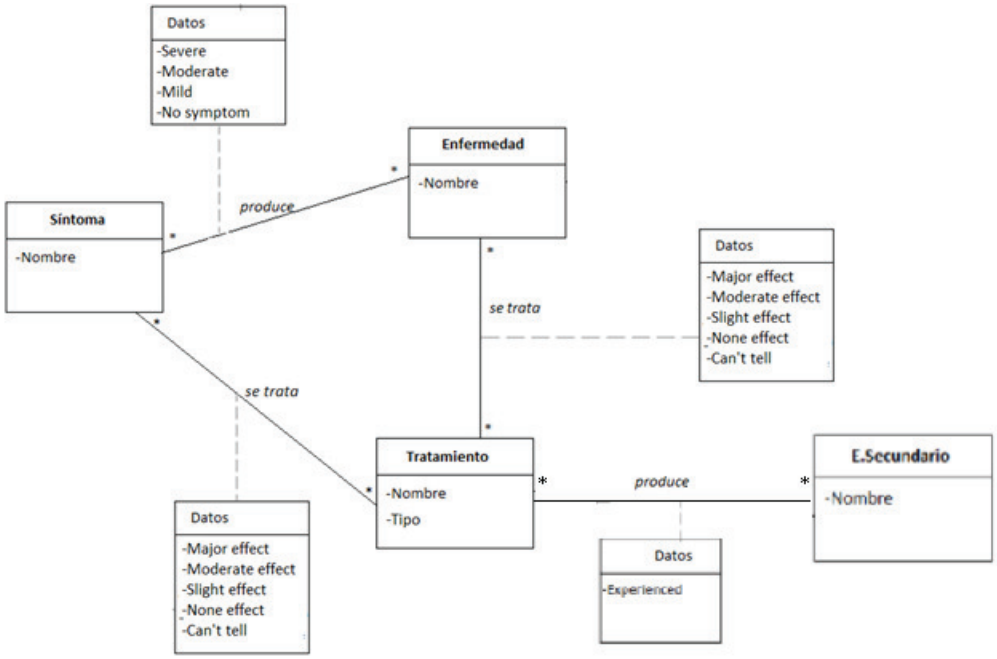


Figura 5.5:Modelo de dominio patientslikeme.com

5.1.2. Detalles de implementación:

A continuación, se describirá como se ha recogido la información de patientslikeme.com

Como ya se ha dicho antes, patientslikeme ofrece un listado de los tratamientos ordenados por número de reportes (https://www.patientslikeme.com/treatments/browse). El primer programa que se ha implementado ha sido para obtener la lista de urls de tratamientos que se van a obtener. Se ha decidido obtener hasta la página 52 del listado, ya que a partir de ésta los reportes son muy reducidos (10 o menos).

Una vez tenemos las urls de los tratamientos que vamos a procesar, hay que analizar el código HTML de los tratamientos para saber en qué parte del código está la información que se desea extraer. Una vez analizado, utilizando BeautifulSoup se accede fácilmente a cada sección. En cada página de tratamiento, se puede obtener tanto relaciones con enfermedades como con efectos secundarios. Tal y como se aprecia en la siguiente figura sacada del tratamiento "Vitamin D" (https://www.patientslikeme.com/treatments/show/322):

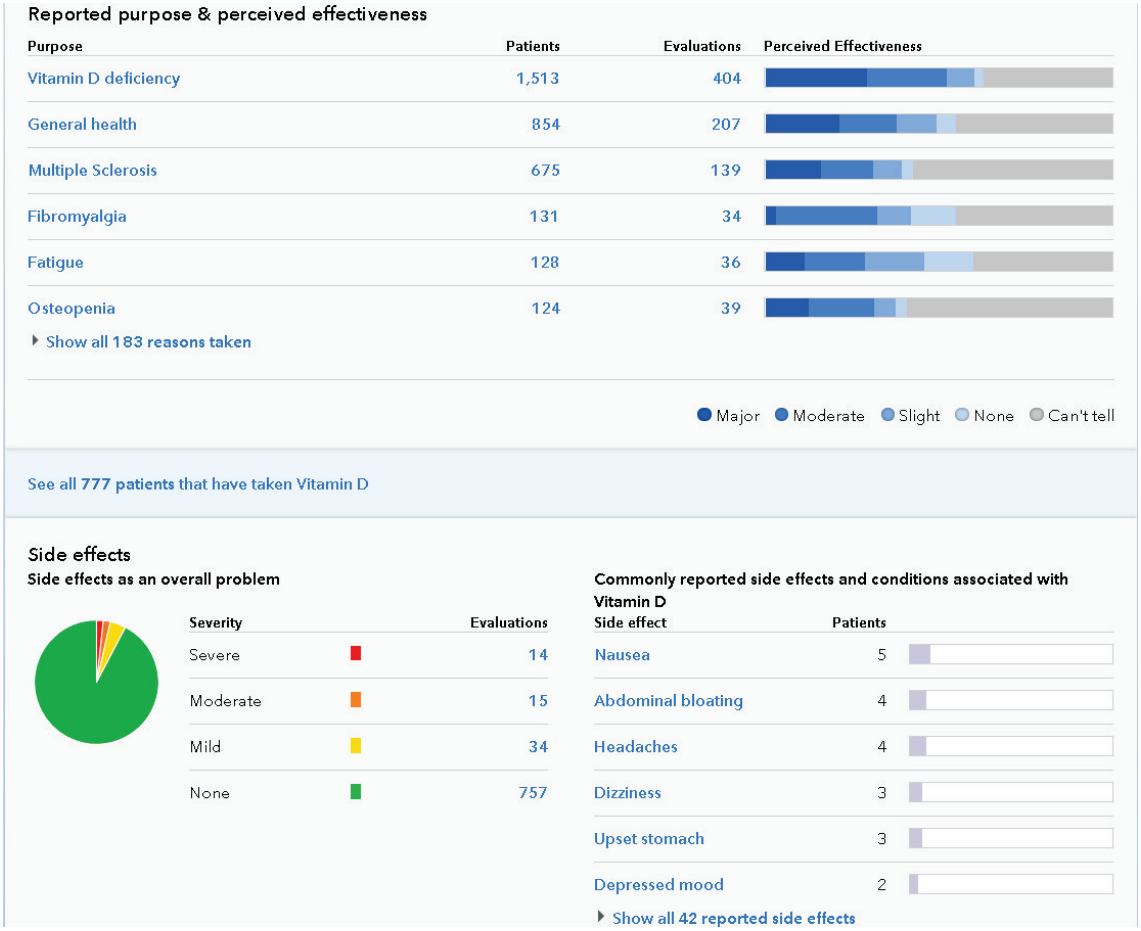


Figura 5.6: Ejemplo página de tratamiento plm

Lo siguiente que se va a realizar es un recorrido por todas las urls que tenemos, imprimiendo en un documento toda la información de cada tratamiento con sus correspondientes eficacias para su respectiva enfermedad por línea. También se guardará en otro fichero la información de los efectos secundarios. En concreto se ha recogido en cada línea el nombre del tratamiento, para qué enfermedad o síntoma se ha usado, qué tipo de tratamiento es, y si para lo que se está usando es una enfermedad o un síntoma, y el número de reportes de cada grado de eficacia tal y como se puede ver en la siguiente imagen anterior. En cuanto a los efectos secundarios, la información es más escasa. Solamente se guardará por línea el tratamiento, su efecto secundario, y el número de reportes.

Por ahora se recogen las escalas tal y como vienen en la página, posteriormente se procederá al reajuste.

Siguiendo con el ejemplo de "Vitamin D", se muestra como es una línea del resultado tras ejecutar el código, la correspondiente a la primera enfermedad "Vitamin D deficiency"

Vitamin D-,-Supplements-,-Vitamin D deficiency-,-condition-,-See 113 evaluations from 98 patients with major perceived effectiveness-,-See 85 evaluations from 74 patients with moderate perceived effectiveness-,-See 31 evaluations from 28 patients with slight perceived effectiveness-,-See 10 evaluations from 9 patients with none perceived effectiveness-,-See 141 evaluations from 114 patients with unknown perceived effectiveness

El primer elemento es el tratamiento, el segundo el tipo, el tercero la enfermedad o síntoma, el cuarto especifica si es enfermedad (condition) o síntoma, y los siguientes son todos los grados de eficacia

Seguido, se muestra otro ejemplo de una línea resultado del fichero de efectos secundarios:

Duloxetine-,-Dry mouth-,-155

Por otro lado, en cuanto a la información de las enfermedades y sus síntomas, no hay ningún tipo de listado. Por suerte, las urls de estas enfermedades siguen el mismo patrón: "https://www.patientslikeme.com/conditions/X" siendo X un número entero y positivo. Manualmente se comprobó hasta qué número existían enfermedades, y se verificó que a partir de 2900 no hay más información. Así que se va a recorrer todas las urls desde X=1 hasta X=2900 y recoger toda la información de cada una, si hay. Realmente se podía haber seguido este mismo procedimiento para los tratamientos ya que sus urls siguen un patrón parecido, pero como se disponía de un listado ordenado, se podía filtrar mejor los resultados.

A continuación, se muestra una imagen de la estructura de la sección de síntomas de una de estas urls de enfermedades, en concreto la de "Lung Cancer" (https://www.patientslikeme.com/conditions/66):

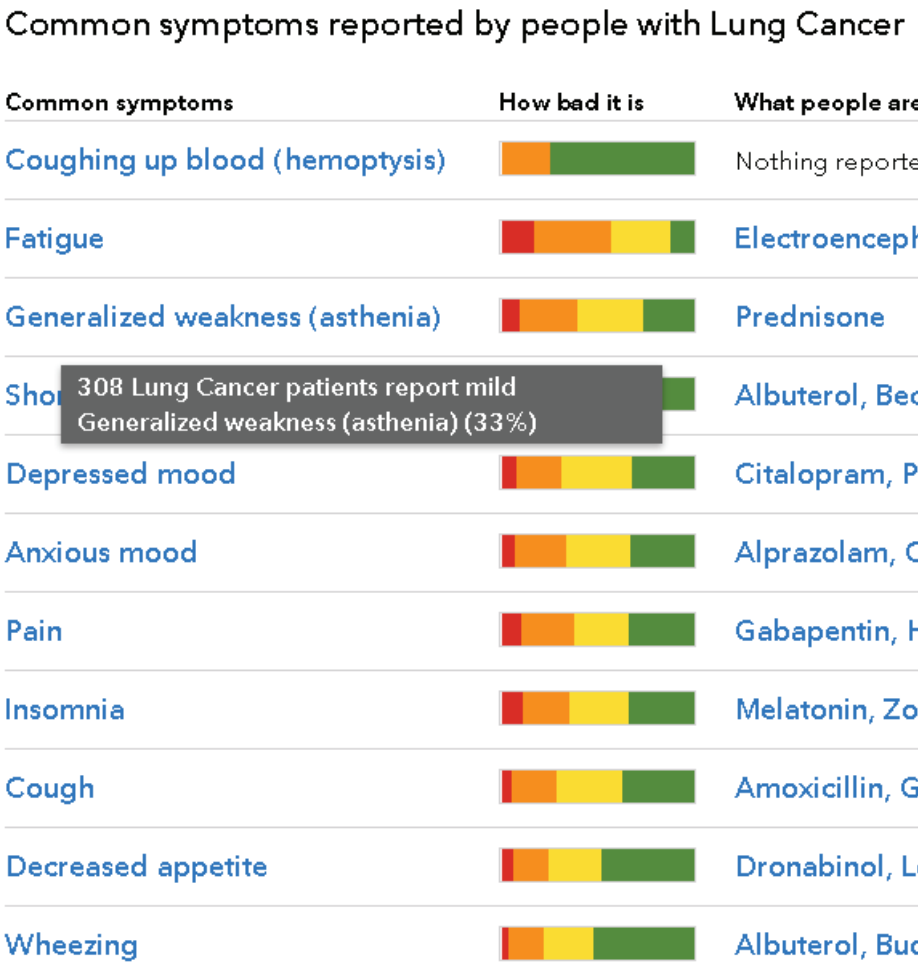


Figura 5.7: Ejemplo síntomas de enfermedad plm

Por cada url de enfermedad, se recogerá la enfermedad, con su síntoma y todos los grados de severidad.

A continuación, se muestra una línea del fichero resultado de síntomas. Siguiendo con el ejemplo se muestra de la enfermedad “Lung Cancer” de la imagen anterior, con el segundo síntoma “Fatigue”:

Lung Cancer-,-Fatigue-,-160 Lung Cancer patients report severe fatigue (18%)-,-352 Lung Cancer patients report moderate fatigue (39%)-,-288 Lung Cancer patients report mild fatigue (32%)-,-111 Lung Cancer patients report no fatigue (12%)-,-

El primer elemento corresponde a la enfermedad, el segundo al síntoma, y el resto los distintos grados de severidad.

Debido al gran número de páginas que hay que procesar, la ejecución de este prototipo puede llegar a durar hasta 2 horas.

5.2. Prototipo 2: Volcado de la información a la ontología.

5.2.1 Análisis y diseño:

Para este prototipo se recoge la información de los ficheros resultado del primer prototipo y se vuelcan en una ontología.

Diagrama de clases del prototipo:

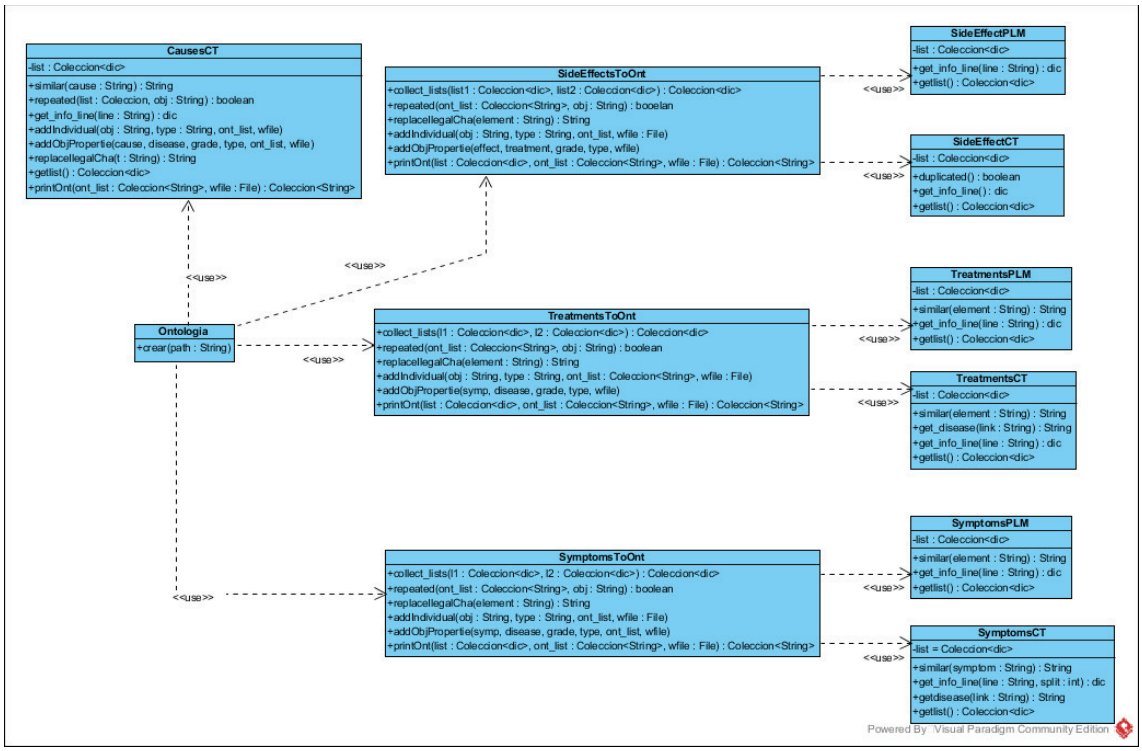


Figura 5.8: Diagrama de clases Ontología

En este apartado de detallará como se ha creado la estructura de la ontología, usando Protégé.

Una vez la información ha sido recogida y analizada, hay que diseñar el modelo de dominio de la ontología.

Hay que tener en cuenta la diferencia entre los grados de severidad de las escalas de las relaciones Enfermedad-Tratamiento y Enfermedad-Síntoma en ambas páginas. Debido a estas diferencias, ha sido necesario reajustar las escalas para obtener una única escala para la ontología. Las siguientes imágenes muestran como ha sido este reajuste. En cuando a los efectos secundarios, como de patientslikeme sólo se obtienen reportes positivos de efectos secundarios, la escala negativa tendrá cero reportes.

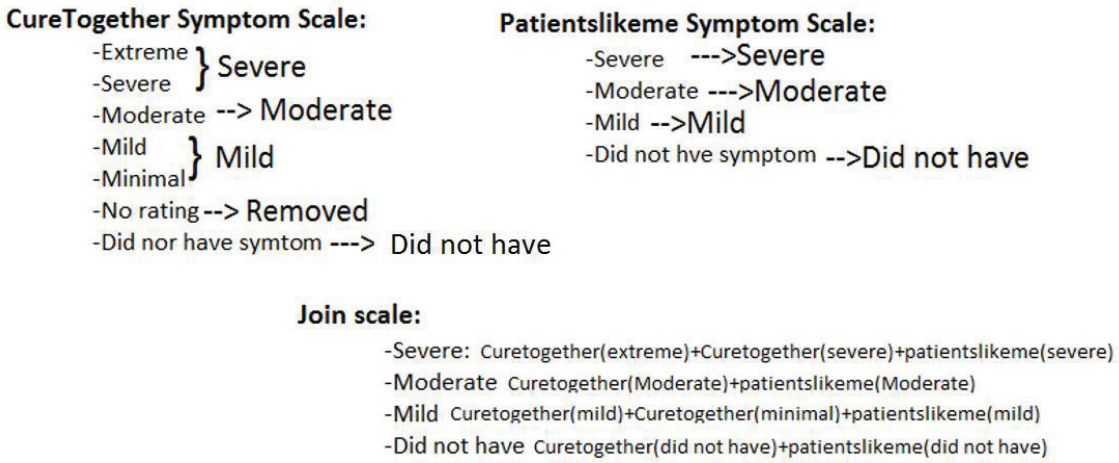


Figura 5.9: Reajuste de escalas síntomas

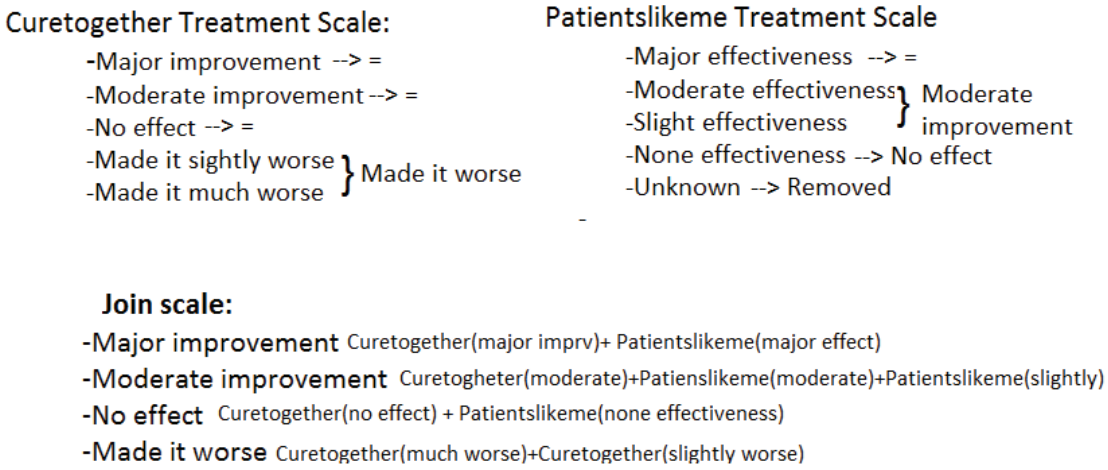


Figura 5.10: Reajuste de escalas tratamientos

Por otra parte, patientslikeme relaciona tratamientos tanto con síntomas como con enfermedades, mientras que en curetogether sólo los relaciona con enfermedades. Este dato hay que tenerlo en cuenta a la hora de diseñar la ontología correctamente.

Por último, patientslikeme asocia cada tratamiento a un tipo, por ejemplo, “prescription drugs”, “exercises” … Por lo que se crearán subclases de la clase tratamiento, una por cada tipo existente más una extra “Unknown” para los tratamientos de curetogether, ya que estos no tienen asociado ningún tipo.

Tras completar, el reajuste de escalas y las diferencias de relaciones, el modelo de dominio resultante es el siguiente:

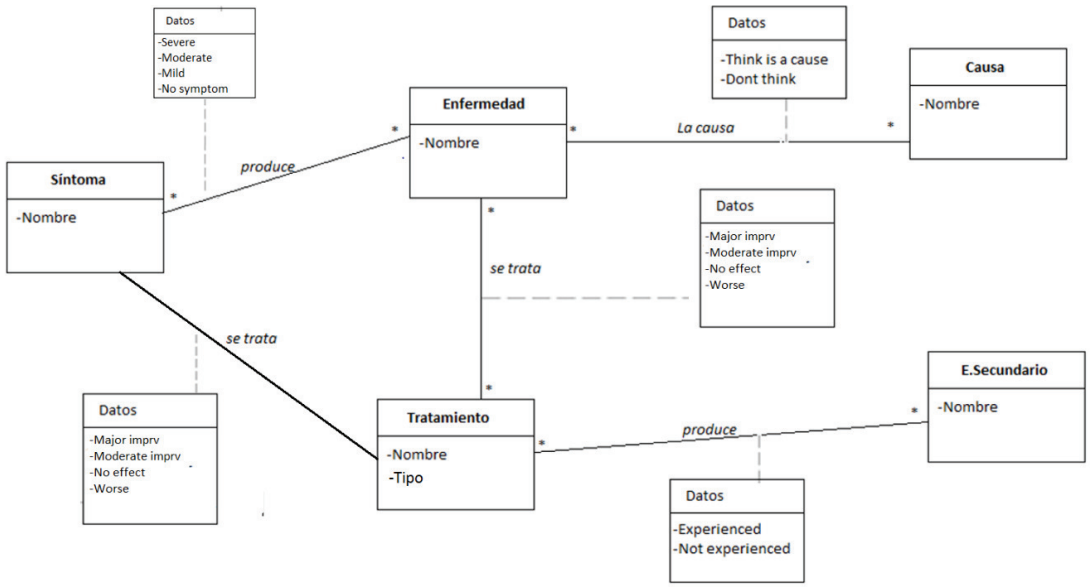


Figura 5.11: Modelo de dominio ontología

El siguiente paso es adaptar la estructura del modelo de dominio a una ontología. Usando la aplicación Protégé 5.0 se crearán los elementos necesarios de la estructura de la ontología. Lo primero que se necesita, es dar un nombre a la ontología. El nombre que se le ha asignado es: *medical_ont*

Cada entidad será representada como una clase (Tratamiento, Enfermedad…), y cada tipo de tratamiento (Exercise, Prescription Drugs) como una subclase de la clase Tratamiento.

Cada relación entre entidades será representada como una Object Propertie. Aquí surge un problema, ya que para cada relación tenemos diferentes grados de severidad y un número de reportes para cada una (severe, moderate…), pero las Object properties no permiten definir atributos, por lo que no hay forma de indicar cuantas evaluaciones hay de cada grado (34 severe, 59 moderate…) únicamente podemos definir la relación. Por eso, relacionar simplemente un síntoma con una enfermedad no va a ser posible. Como solución a este problema, se ha creado una clase intermedia para cada relación (Enfermedad-sintoma, tratamiento-enfermedad…) y una subclase por cada grado de severidad de cada relación. Por lo tanto, para representar un grado de severidad entre dos instancias, será necesario crear una tercera instancia de esta nueva clase intermedia para representar la relación. Estas instancias intermedias tendrán un atributo que indicará el número de evaluaciones que tiene. Por motivos de espacio, si una escala tiene menos de 10 reportes, no se añadirá a la ontología. La siguiente figura muestra un ejemplo de lo explicado:

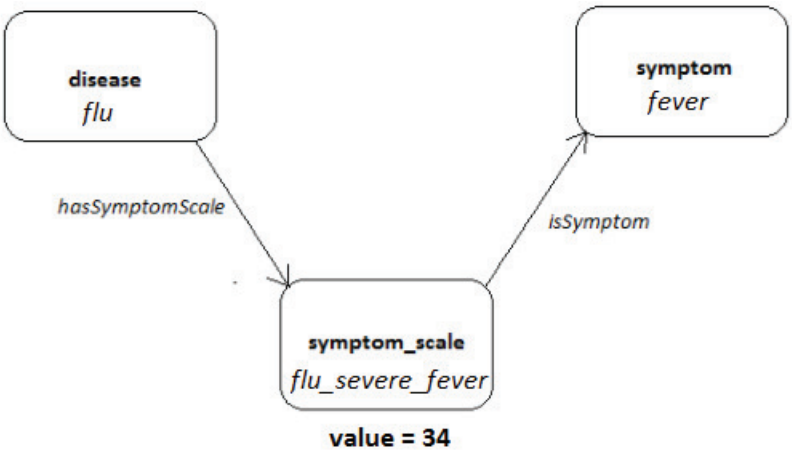


Figura 5.12: Ejemplo relación ontología

Siguiendo con las relaciones, para cada Object propertie hay que definir un dominio (domain) y un rango (range) que representan los nodos de la relación. Siguiendo con el ejemplo anterior, el dominio de *hasSymptomScale* sería *disease*, y el rango sería *symptom_scale*. Este procedimiento es similar para todas las relaciones.

Y ahora, se muestra una imagen de las Object properties generadas (dos por cada relación entre entidades):

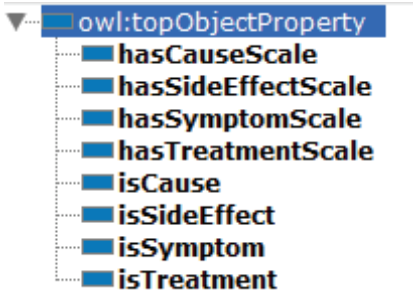


Figura 5.13: Object Properties ontología

Ese *value* que tendrá cada instancia, que se mostraba en la figura 5.12 de la clase de grado de severidad, será un Data Propertie de tipo *xsd:int*.

A continuación, se muestra en una imagen las clases y subclases generadas:

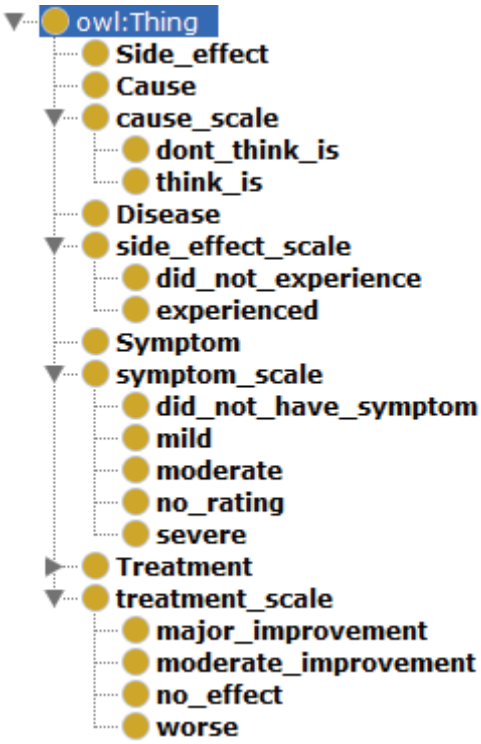


Figura 5.14: Clases ontología

Como se puede observar, para cada clase de escalas, se han creado tantas subclases como grados de severidad tiene.

Por último, cada enfermedad, síntoma… etc. será una instancia de la ontología *(Individuals)*. Para cada grado de severidad de cada relación entre dos instancias, también habrá que crear la instancia de la clase intermedia, como se ha explicado anteriormente.

Finalmente, se muestra un gráfico representando las relaciones de clases de la ontología creada con Protégé. Posteriormente se añadirán las distintas enfermedades, tratamientos, síntomas, etc de los ficheros del Crawler como instancias. Esta ontología se guardará como "ont_body" en un fichero owl el cual el programa implementado ampliará con las instancias mencionadas.

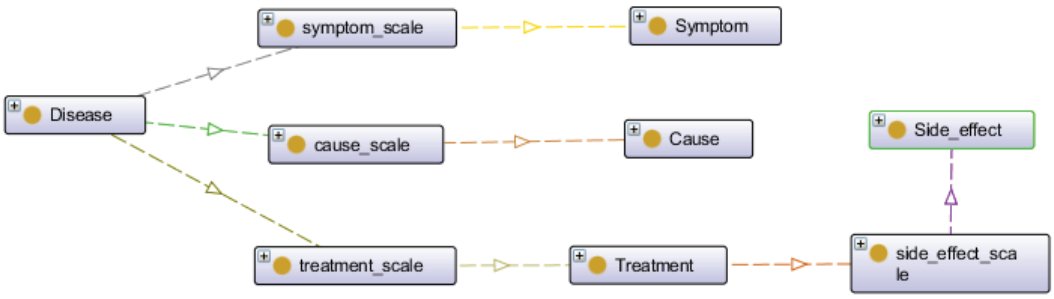


Figura 5.15: Diagrama de relaciones ontología

5.2.2 Detalles de implementación:

Una vez completada la estructura de la ontología, con Protégé, se procede a volcar la información que tenemos de las webs. Una vez finalizado el "crawling" del prototipo 1 contamos con la siguiente información:

- Curetogether
    - Tratamientos (relacionados con enfermedades)
    - Síntomas (relacionados con enfermedades)
    - Causas (relacionados con enfermedades)
    - Efectos secundarios (relacionados con tratamientos)
- Patientslikeme
    - Tratamientos (relacionados con enfermedades o síntomas)
    - Síntomas (relacionados con enfermedades)
    - Efectos secundarios (relacionados con tratamientos)

En este paso, se va a usar la biblioteca de Python "re" (Python Software Foundation, 2012) que permite trabajar con expresiones regulares, para poder obtener la información que necesitemos de los ficheros .csv. Además, aquí es donde se procederá a reajustar las escalas.

Para cada fichero, se creará una clase que gestionará la recogida de información y volcado de la ontología de ese fichero. Finalmente, se creará una clase que gestionará todas las demás.

Ya que hay elementos que pueden aparecer más de una vez, habrá que controlar no añadir dos veces una misma instancia a la ontología, para ello simplemente se creará lista en la que se añadirán los elementos que se van volcando en ésta, y nos aseguramos de no añadir instancias repetidas. También ocurre que dos elementos son lo mismo, pero están escritos de manera diferente, por ejemplo: *adrenal fatigue, Adrenal fatigue, adrenal-fatigue.* Debido a esto, se necesitará crear un método que compruebe si dos elementos son iguales, pero escritos diferentes. El funcionamiento de este, consiste en compararlos habiéndoles eliminado espacios, guiones, y poniéndolo en minúsculas. Aún con esto, sigue habiendo elementos que tienen el mismo significado, pero su escritura es bastante más diferente. No se ha realizado ningún estudio para el control de este tipo de elementos.

Para el volcado de causas, como solamente hay un fichero con información (de curetogether), se puede simplemente recoger la información y volcarla en la ontología. Sin embargo, con la efectividad de tratamientos, síntomas y efectos secundarios, al haberlos recogido de dos páginas diferentes, va a ser necesario realizar un preproceso mayor: por un lado, tal y como se ha explicado anteriormente, hay que reajustar las escalas, ya que cada página tiene la suya. Por otro lado, ya controlamos que no se añadan dos instancias iguales, pero no dos relaciones iguales. Al haber recogido datos de dos páginas, puede darse el caso de que la misma relación esté en ambas páginas, pero en la ontología solo deberá aparecer una, es decir, si de curetogether se ha sacado que el síntoma *s--fever* tiene 5 reportes de grado *severe* para la enfermedad *d--flu*, y por otro lado en patientslikeme está esa misma relación, pero con 15 reportes, en la ontología habrá que añadir 20 reportes. Por eso se necesitará tres clases extra para gestionar estas complicaciones a la hora de pasar la información a la ontología.

Lo primero que se ha realizado es eliminar, o cambiar una serie de caracteres inválidos, que causaban problemas a la hora de crear las instancias de la ontología. La siguiente tabla muestra los cambios que se han hecho en los elementos:

Antes	Después
espacio	_
&	and
%	pct
<	less_than
>	more_than
#, [,], {, }, /,\,’	

Tabla 5.1: Cambio de caracteres inválidos

Hay que tener en cuenta que una misma palabra puede haber sido reportada en diferentes tipos, por ejemplo, ‘headache’ puede haber sido reportado como síntoma y también como efecto secundario. Sin embargo, no se puede albergar dos instancias con el mismo nombre, aunque sean de distintas clases. Para poder albergar un mismo término en diferentes clases, se añadirá al comienzo del nombre la etiqueta ‘x--‘ donde x será distinta dependiendo de a qué clase vaya a pertenecer ese término. Por ejemplo, si estamos tratando ‘headache’ como enfermedad (disease) se añadirá a la ontología como ‘d--headache’, sin embargo, si fuera como efecto secundario (side effect) se añadiría como ‘se--headache’. De esta forma podemos tener mismos términos en clases distintas.

Cabe mencionar, que los ficheros de curetogether contenían líneas en los que algunos elementos estaban entre comillas (“”). Estas comillas han sido eliminadas antes de procesar las líneas ya que producían errores a la hora de ejecutar las expresiones regulares. Por otra parte, como ya se ha explicado, se ha usado “-,-“ para dividir los elementos en los ficheros de patientslikeme, pero en curetogether estaban separados por comas. Esto dificultaba el realizar el split de los elementos de algún fichero, ya que algún elemento podría contener comas y sería dividido de forma incorrecta o detectados incorrectamente por una expresión regular en las que aparecen comas. Por suerte, había una diferencia entre las comas que dividían elementos y las comas que formaban parte de ellos: las que dividían elementos no tenían ningún espacio entre ellas, y las que no usaban un espacio después de la coma, por lo que se cambió todas las comas que pudieran contener los elementos por guiones para que no entorpecieran la ejecución de las expresiones regulares y división de elementos.

- **Obtención de causas del fichero de curetogether**

En el fichero de causas de curetogether, por cada línea nos muestra cuantos pacientes piensan que “algo” causa su enfermedad, por ejemplo:

49 out of 439 respondents (11%) think Oxalates is a cause of their Vulvodynia,Oxalates,http://curetogether.com/vulvodynia/causes/

En este ejemplo 49 personas han reportado que *Oxalates* es una causa de *Vulvodynia* .Varias líneas de este csv tenían datos sin sentido, en las que no había número de positivos, y empezaba la línea de tal forma: ” *out of 0 (0%)*”. Estas líneas se han ignorado.

Para obtener información de este csv, se ha usado la siguiente expresión regular:

```
p2 = re.compile(
'(\d+) out of (\d+) respondents [(]\d*[%][)] think (.*) is a cause of their (.*?),\w')
Positives     total                                cause                 disease
```

Con éste patrón, ya tenemos toda la información que necesitamos: los positivos (que piensan que es una causa) los totales, la causa y la enfermedad.

- **Obtención de efectos secundarios del fichero curetogether:**

Cada línea de este fichero contiene efectos secundarios de un tratamiento junto con un grado de aparición (cuanta gente piensa que lo ha sufrido).

31 out of 108 respondents (29%) experienced diarrhea with Doxycycline,15 out of 108 respondents (14%) experienced itching of the rectum or vagina with Doxycycline,9 out of 107 respondents (8%) experienced sore mouth with Doxycycline,4 out of 14 respondents (29%) experienced Low Fever with Doxycycline,1 out of 17 respondents (6%) experienced cough with Doxycycline,Doxycycline,http://curetogether.com/acne/side-effects/

Como se puede observar, en cada elemento que contiene el grado de aparición de un efecto secundario aparece toda la información necesaria que se desea recoger:

31 out of 108 respondents (29%) experienced diarrhea with Doxycycline
Positives Total side effect Treatment

Como vemos, se puede obtener toda la información con este único elemento, se va a separar la línea por comas (,) y se van a tratar los elementos por separado usando la expresión regular de debajo. Es posible que algún elemento de la línea contenga comas. Estas comas se han cambiado por guiones (-) para que el split se realice de forma correcta. Los últimos elementos no nos interesan, ya que esa información ya la hemos recogido.

La expresión regular que se ha implementado para recoger la información se muestra a continuación:

```
p1 = re.compile(
'(\d+) out of (\d+) respondents [(]\d*[%][)] experienced (.*) with (.*)')
Positives    Total                                  side effect treatment
```

Tras aplicar la expresión regular, al igual que con las causas, guardamos los datos en un diccionario, el cual se guardará en una lista con todos los diccionarios de cada línea. Antes de crear el diccionario, se comprueba que no está ya en la lista, ya que es el único fichero donde puede haber relaciones repetidas.

- **Obtención de efectos secundarios del fichero de patientslikeme:**

Las líneas de este fichero son bastante simples, como ya se ha visto anteriormente. Ni ha sido necesario usar la librería "re".

Duloxetine-,-Dry mouth-,-155

El primer elemento corresponde con el tratamiento. El segundo es el efecto secundario. El número representa los reportes positivos. No existen reportes negativos, como en curetogether, por lo que solo se recogerán los positivos.

- **Obtención de tratamientos del fichero de curetogether:**

Este csv ha sido el más complicado, por cada línea nos muestra cómo afecta un tratamiento a los pacientes en cada grado de severidad, por ejemplo:

2 people thought Avoid certain people made their PTSD much worse,3 people thought Avoid certain people made their PTSD slightly worse,11 people thought Avoid certain

people had no effect on their PTSD,33 people experienced a moderate improvement after trying Avoid certain people,19 people experienced a major improvement after trying Avoid certain people,Avoid certain people,http://curetogether.com/ptsd/treatments/

Para obtener información de este csv, se han necesitado 3 expresiones regulares para obtener los grados de eficacia diferentes, ya que se expresa de diferente forma para cada grado de severidad (menos para major-moderate improvement y sightly-much worse). En cuando al tratamiento y la enfermedad, se puede obtener fácilmente ya que aparecen en los dos últimos elementos de la línea (la enfermedad para la que se trata aparece dentro del link) la cual se divide por comas (,):

```
#Para much o sightly worse
p1 = '(\d+) people thought (.*?) made their (.*?) (\w+)
     reportes          tratamiento         enf  palabra(much o sightly))
#Para no effect
p2 = '(\d+) people thought (.*?) had no effect on their (.*)'
     reportes          tratamiento                    enfermedad
#Para major y moderate improvement
 p3 = '(\d+) people experienced a (\w+) '
     reportes                    palabra (major o moderate)
```

Una vez comprobados todos los elementos de la línea, se realiza el ajuste de escalas, sumando los reportes de las escalas que se van a juntar, tal y como se ha explicado anteriormente.

Finalemente se guarda la información en un diccionario y se añade a la lista de diccionarios.

- **Obtención de tratamientos del fichero de patientslikeme**:

Al igual que en las líneas de curetogether, en este fichero se muestra los grados de eficacia de un tratamiento para una enfermedad. A continuación, se muestra un ejemplo:

Gabapentin-,-Prescription Drugs-,-Nerve pain (neuralgia)-,-symptom-,-See 223 evaluations from 182 patients with major perceived effectiveness-,-See 364 evaluations from 312 patients with moderate perceived effectiveness-,-See 212 evaluations from 185 patients with slight perceived effectiveness-,-See 86 evaluations from 75 patients with none perceived effectiveness-,-See 66 evaluations from 59 patients with unknown perceived effectiveness

Se dividen los elementos por "-,-". Los cuatro primeros elementos son el tratamiento, que tipo de tratamiento es, para qué se ha usado, y si para lo que se ha usado es síntoma o enfermedad (en este caso síntoma). El resto son grados de eficacia. Para recoger la información de cada elementos se crea la siguiente expresión regular:

```
'See \d+ evaluations from (\d+) patients with (\w+)
                          num reportes          grado de eficacia
```

Tras obtener toda la información, se reajustarán las escalas, sumando la información de las que se vayan a unir, o ignorando las que no. Se añade a un diccionario y se añade a la lista de diccionarios

- **Obtención de síntomas de curetogether:**

Las líneas de cada fichero muestran grados de aparición de un síntoma en una enfermedad tal y como se muestra a continuación:

39 people reported minimal Hoarseness,62 people reported mild Hoarseness,65 people reported moderate Hoarseness,16 people reported severe Hoarseness,9 people reported

extreme Hoarseness,40 people reported having Hoarseness but did not provide a severity rating,421 people reported not having Hoarseness,Hoarseness,http://curetogether.com/acid-reflux/symptoms/;;;;;;;;

Para cada línea, se dividirán los elementos por comas (tras haber sustituido las posibles comas dentro de los elementos por guiones). Cada elemento muestra información sobre un grado de aparición. Los dos últimos son de donde se recogerán el síntoma y la enfermedad (dentro del link). Para obtener la información de cada grado, se usará la siguiente expresión regular:

```
'(\d+) people reported (\w*)\s'
 Reportes               grado
```

Tras obtener toda la información, se reajustarán las escalas, sumando la información de las que se vayan a unir, o ignorando las que no. Se añade a un diccionario y se añade a la lista de diccionarios

- **Obtención de síntomas de patientslikeme:**

La estructura de cada línea de este fichero es similar a la de curetogether:

Epilepsy-,-Problems concentrating-,-769 Epilepsy patients report severe Problems concentrating (17%)-,-1,503 Epilepsy patients report moderate Problems concentrating (33%)-,-1,387 Epilepsy patients report mild Problems concentrating (31%)-,-766 Epilepsy patients report no Problems concentrating (17%)-,-

Tras separar los elementos por "-,-", de los dos primeros se obtienen la enfermedad y el síntoma. El resto son grados de severidad, de los cuales se usará la siguiente expresión regular para obtener información:

```
(\d+) .* patients report (\w+)
reportes                 grado
```

Tras obtener toda la información, se reajustarán las escalas, sumando la información de las que se vayan a unir, o ignorando las que no. Se añade a un diccionario y se añade a la lista de diccionarios

- **Volcar información a la ontología:**

Una vez se han recogido los diccionarios de un tipo (tratamientos, síntomas, efectos y causas) y de juntar la información de las listas de los resultados de las dos webs (excepto en la causa que solo hay de curetogether) se van a añadir a la ontología. Es decir, la información se irá añadiendo en bloques, primero se recoge toda la de, por ejemplo, síntomas de los ficheros, se comprueba que no haya relaciones repetidas, y se añade; luego lo mismo con las enfermedades, con los tratamientos y con los efectos.

Se estudiaron posibles librerías para Python que realizaran las operaciones necesarias para añadir las instancias y las relaciones a una ontología, pero como no se encontró ninguna que satisficiera las necesidades, finalmente se decidió imprimir en el fichero donde se creó la estructura de la ontología con Protété toda la información obtenida usando la función *write*, usando la sintaxis OWL/XML.

Para añadir cada instancia, el código que se imprime es el siguiente:

```
<Declaration>
        <NamedIndividual IRI="#' + obj + '"/>
```

```
</Declaration>
<ClassAssertion>
        <Class IRI="#'+ type +'"/>
        <NamedIndividual IRI="#'+ obj + '"/>
</ClassAssertion>\n'
```

En la primera línea se declara la instancia (nombre = obj).

En la cuarta, se especifica de que tipo es la instancia (la instancia obj es del tipo type), por ejemplo, Disease

Para cada relación, el código es un poco más amplio, y se muestra a continuación:

Primero se declara la instancia intermedia explicada anteriormente:

```
<Declaration>
        <NamedIndividual IRI="#' + obj + '"/>
</Declaration>
```

Se le asigna la clase (la clase correspondiente a la escala en cuestión)

```
<ClassAssertion>
        <Class IRI="#' + type + '"/>
        <NamedIndividual IRI="#' + obj + '"/>
</ClassAssertion>
```

Se añaden las dos relaciones necesarias, especificando los dos nodos de la relación(dominio y rango). En este case se muestra un ejemplo para la relación Enfermedad-síntoma.

```
<ObjectPropertyAssertion>
        <ObjectProperty IRI="#hasSideEffectScale"/>
        <NamedIndividual IRI="#' + element1 + '"/>(nodo1, dominio)
        <NamedIndividual IRI="#' + obj + '"/>(nodo2, rango)
</ObjectPropertyAssertion>

<ObjectPropertyAssertion>
        <ObjectProperty IRI="#isSideEffect"/>
        <NamedIndividual IRI="#' + obj + '"/>(nodo1, dominio)
        <NamedIndividual IRI="#' + element2 + '"/>(nodo2, rango)
</ObjectPropertyAssertion>
```

Y finalmente se añade el valor de la relación

```
<DataPropertyAssertion>
        <DataProperty IRI="#value"/>
        <NamedIndividual IRI="#' + obj + '"/>
        <Literal datatypeIRI="http://www.w3.org/2001/XMLSchema#integer">
        '+str(grade) + '</Literal>
</DataPropertyAssertion>
```

A continuación, se muestra un ejemplo de lo explicado. La relación de ejemplo es la siguiente:

10 flu** patients reported **severe fever

Se trata de una relación de síntoma-enfermedad. *Flu* es la enfermedad, y *fever* el síntoma. El grado de la relación es *severe*, y tiene *10* reportes.

Primero se añaden las dos instancias, flu y severe. Recordar que se añaden con una etiqueta (d--)

Añadimos **flu**, clase Disease:

```
<Declaration>
        <NamedIndividual IRI="#d--flu"/>
</Declaration>
```

```
<ClassAssertion>
        <Class IRI="#Disease"/>
        <NamedIndividual IRI="#d--flu”/>
</ClassAssertion)
```

Añadimos **fever**, clase Symptom:

```
<Declaration>
        <NamedIndividual IRI="#s--fever"/>
</Declaration>

<ClassAssertion>
        <Class IRI="#Symptom"/>
        <NamedIndividual IRI="#s--fever”/>
</ClassAssertion)
```

Añadimos la relación **flu_severe_fever**, con **10 reportes**:

Primero se declara la instancia intermedia explicada anteriormente y se le asigna la clase(severe):

```
<Declaration>
        <NamedIndividual IRI="#d--flu_severe_s--symptom"/>
</Declaration>
<ClassAssertion>
        <Class IRI="#severe"/>
        <NamedIndividual IRI="# d--flu_severe_s--symptom "/>
</ClassAssertion>
```

Se añaden las dos relaciones necesarias:

```
<ObjectPropertyAssertion>
        <ObjectProperty IRI="#hasSymptomScale"/>
        <NamedIndividual IRI="#d--flu"/>
        <NamedIndividual IRI="#d—flu_severe_s--fever"/>
</ObjectPropertyAssertion>

<ObjectPropertyAssertion>
        <ObjectProperty IRI="#isSymptom"/>
        <NamedIndividual IRI="# d—flu_severe_s--fever "/>
        <NamedIndividual IRI="#s--fever"/>
</ObjectPropertyAssertion>\n')
```

Y finalmente se añade el valor de la relación a la instancia intermedia:

```
<DataPropertyAssertion>
        <DataProperty IRI="#value"/>
        <NamedIndividual IRI="# d—flu_severe_s--fever "/>
        <Literal datatypeIRI="http://www.w3.org/2001/XMLSchema#integer"> 10
        </Literal>
</DataPropertyAssertion>\n')
```

5.3 Prototipos 3 y 4: Reconocimiento de conceptos en textos y dar información

5.3.1. Análisis y diseño

Una vez se ha volcado toda la información en la ontología, el siguiente paso consistirá en implementar un detector que sea capaz de identificar en un texto los elementos que están albergados en la ontología. Y finalmente, con el resultado de éste detector, se mostrará información acerca de los términos que hayan sido encontrados.

El diagrama de clases de estos dos prototipos es el siguiente:

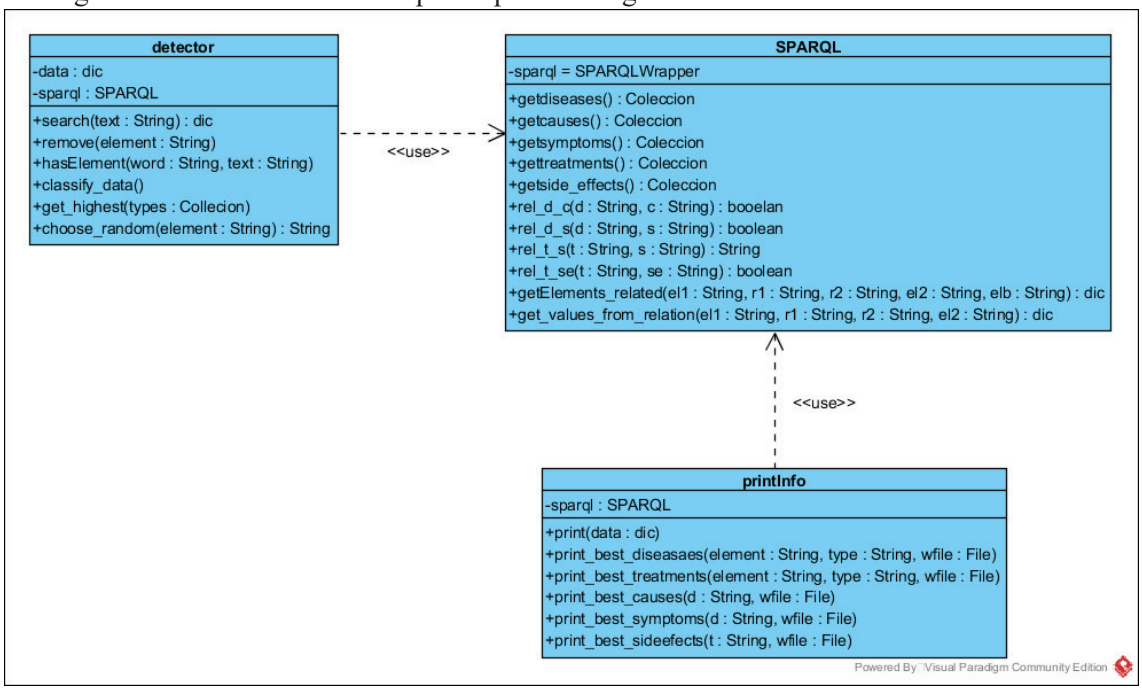


Figura 5.16: Diagrama de clases prototipos 3 y 4

5.3.2 Detalles de implementación

El funcionamiento de estos prototipos se puede dividir en 3 partes principales:

- Obtener elementos de la ontología
- Comparar cada elemento con el texto de entrada y determinar de qué clase es.
- Mostrar información de interés a partir de los elementos reconocidos.

Al ejecutar el programa, se comprobará para cada elemento de la ontología si aparece en el texto de entrada. Para ello habrá que quitarle la etiqueta que se le añadió para poder albergarlo en más de una clase, y cambiar la separación en barras bajas (_) de los elementos por espacios, ya que los textos estarán separados por espacios. Una vez los elementos estén listos para ser comparados, se usará la librería "re" de nuevo para crear un patrón para comprobar si ese elemento está en el texto. El patrón es el siguiente:

```
'.*\W('+word+')\W.*'
```

Para recoger los elementos de la ontología, se ha usado uso de la librería SPARQLWrapper, la cual permite realizar consultas SPARQL. Aquí han surgido dos inconvenientes. El primero es

que el fichero de la ontología que se ha generado, está en formato OWL/XML, y esta librería sólo admite archivos en formato RDF/XML. EL segundo, es que la librería no permite recoger una ontología desde un fichero, si no que desde una URL. A continuación, se explica cómo se han solucionado estos dos inconvenientes:

1. Protègè, permite cargar y guardar ontologías en diferentes formatos. Se puede cargar en formato OWL/XML y guardar en RDF/XML. Para realizar esta función de manera automática si, se ha usado la API "owlapi", que es la misma que usa Protègè. Se intentó programarlo desde Python, pero las funcionalidades de esta API para este lenguaje son muy limitadas y no se consiguió, por lo que fue necesario implementarlo desde Java, ya que fue diseñada para este lenguaje. Una vez implementado el conversor en java, se ha generado el ejecutable JAR, y desde Python se ha llamado a éste, pasándole como parámetro el directorio del fichero OWL de la ontología para transformarlo en formato RDF/XML.
2. Para poder cargar la ontología usando SPARQLWrapper es necesario hacerlo desde una url. Para ello, se ha decidido albergar la ontología en un servidor local. Para crear un servidor, se ha usado al API de Apache Jena Fuseki, que permite cargar ontologías desde ficheros en un servidor local, siendo por defecto "localhosl:3030", si este puerto no está ocupado, que no es el caso. Por suerte, esta API acepta ficheros en formato RDF/XML.

   Para iniciar el servidor y cargar la ontología desde Python, se puede conseguir fácilmente ejecutando el siguiente comando:

```
os.system(
"java –jar path_de_jena_fuseki.jar --file path_de_la_ontología
/Data")
```

   Con esto, se consigue iniciar el servidor en localhost:3030, y cargar la ontología en un "dataset" llamado Data.

Una vez completados los dos pasos anteriores, ya se puede empezar a usar SPARQLWrapper.

Los elementos de la ontología se recogerán por clases para compararlos en el texto, siendo necesaria una consulta SPARQL para cada clase. Como ejemplo, se muestra a continuación cómo sería la consulta para recoger las enfermedades (clase Disease):

```
PREFIX: <http://www.semanticweb.org/medical_ont#>
PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#

SELECT ?d WHERE {
      ?d rdf:type :Disease}
```

Al recogerlos por clases, ya se sabe a qué clase pertenece cada elemento reconocido, por lo que a la vez que se van identificando, también se están clasificando. El único problema que puede surgir es con aquellos términos que aparezcan repetidos en diferentes clases. Por ejemplo, si el texto tiene el elemento "*headache*" y en la ontología se tiene "*d--headache*" como enfermedad, y "*s--headache*" como síntoma, este elemento estará tanto en los resultados de enfermedades como en los de síntomas.

Para elegir si clasificar estos elementos repetidos de un tipo u otro, se comprueba cuantos reportes tendría con el resto de términos encontrados en el texto si fuera de cada tipo existente (en el caso de *headache*, como enfermedad y como síntoma), y se clasifica como la clase para la que más reportes haya encontrado (si encuentra 51 reportes siendo enfermedad y 4 siendo síntoma, se clasificaría como enfermedad). En caso de que hubiera el mismo número de reportes en más de un tipo, se elegirá uno de estos aleatoriamente. A continuación, se muestra un texto muy simple de ejemplo y como se clasificarían:

Migraine, nausea, fatigue

El detector va a identificar estos tres términos, ya que los tres aparecen en la ontología. Los tres aparecen repetidos, *migraine* aparece como síntoma, como tratamiento (aunque no tenga mucho sentido) y como enfermedad; *nausea* y *fatigue* aparecen como enfermedad, como síntoma y como efecto secundario. A la hora de aplicar lo explicado anteriormente, para *migraine* se va a obtener un número X de reportes siendo una enfermedad, porque en la ontología está relacionada con *nausea* y *fatigue* como síntomas. Siendo tratamiento o síntoma no se va a encontrar ninguno, ya que no hay relaciones con los otros dos elementos no siendo enfermedad, por lo que se va a clasificar *migraine* como enfermedad, al tener un número mayor de reportes.

Para obtener el número de reportes de una relación entre dos términos, se necesitará implementar tantas consultas SPARQL como tipos de relaciones haya, es decir, una para enfermedad-síntoma, otra para enfermedad-tratamiento, etc. A continuación, se muestra un ejemplo de consulta SPARQL para comprobar obtener los reportes de una enfermedad y un síntoma. Los elementos serán "*d--headache*" y "*s--pain*". Como es una relación enfermedad-síntoma, se usarán las objectProperties "*hasSymptomScale*" y "*isSymptom*":

```
PREFIX : <http://www.semanticweb.org/medical_ont#>
SELECT  SUM(?v) WHERE {
:d--headache :hasSymptomScale ?object.
?object :isSymptom :s—pain.
?object :value ?v}
```

Con esto, se suman los reportes de todos los grados de severidad de la relación encontrados. Si no existe relación (no hay reportes) el resultado será 0.

Finalmente, para generar información sobre los resultados del detector, se ha implementado dos algoritmos, el primero únicamente muestra las relaciones directas que se han encontrado entre los elementos detectados. Este será el primero que se ejecutará, y si el usuario desea obtener información extra, se ejecutará el segundo.

El primer algoritmo, es el siguiente:

```
Para cada enfermedad detectada:
    Para cada tratamiento detectado:
        Mirar si tiene relación con la enfermedad:
        Si la tiene:
            Imprimir grados de relaciones
```

```
Para cada causa detectada:
      Mirar si tiene relación con la enfermedad:
      Si la tiene:
            Imprimir grados de relaciones
Para cada síntoma detectado:
      Mirar si tiene relación con la enfermedad:
      Si la tiene:
            Imprimir grado de relaciones
Para cada efecto secundario detectado:
Para cada tratamiento detectado:
      Mirar si tiene relación.
      Si tiene:
            Imprimir grados de relaciones
```

A continuación, se muestra un ejemplo del resultado de este algoritmo. En la interfaz, se ha introducido, *migraine aspirin nausea*. El detector va a clasificar *migraine* como enfermedad, *aspirin* como tratamiento, y *nausea* como síntoma. Por lo que el algoritmo debería mostrar las relaciones *migraine-aspirin y migraine-nausea*:

Figura 5.17: Ejemplo de resultado

Al pulsar el botón guardar, se guardará la información en un archivo ".txt", en un directorio por defecto

Al pulsar "Mas info" se ejecutará el segundo algoritmo.

El segundo, aparte de mostrar la información del primero, muestra los elementos relevantes de cada término encontrado (si los tiene). Por cada elemento se imprimirá los porcentajes de sus grados de relaciones. El algoritmo es el siguiente:

```
Para cada enfermedad detectada:
        Para cada tratamiento detectado:
                Mirar si tiene relación con la enfermedad:
                Si la tiene:
                        Imprimir grados de relaciones
```

```
        Para cada causa detectada:
                Mirar si tiene relación con la enfermedad:
                Si la tiene:
                        Imprimir grados de relaciones
        Para cada síntoma detectado:
                Mirar si tiene relación con la enfermedad:
                Si la tiene:
                        Imprimir grado de relaciones


        Mostrar tratamientos relevantes (los usuarios han experimentado mejora)
        Mostrar tramientos perjucidiales (han hecho que los usuarios vayan a peor)
        Mostrar causas relevantes
        Mostrar síntomas relevantes


Para cada tratamiento detectado:
        Mostrar efectos secundarios relevantes.
Para cada síntoma detectado:
        Mostrar enfermedades relevantes
Para cada causa detectada:
        Mostrar enfermedades relevantes


Para cada efecto secundario detectado:
        Para cada tratamiento detectado:
                Mirar si tiene relación.
                Si tiene:
                        Imprimir grados de relaciones
        Mostrar tratamientos relevantes
```

Para determinar si un elemento es relevante, se ha obtenido un coeficiente para cada relación con otro elemento posible. Este coeficiente se obtiene calculando el porcentaje de reportes de los grados de relación más altos. Por ejemplo, para las relaciones enfermedad-síntoma (grados *severe, moderate, mild, dont have*) se ha obtenido el porcentaje de *severe* + *moderate*. Se mostrarán únicamente los resultados con un coeficiente superior a **70%,** a excepción de los efectos secundarios y los tratamientos que provocan empeoramiento, ya que al ser algo más perjudicial, se ha decidido establecer el umbral en **50%**.

A continuación, se muestra parcialmente, el resultado de este algoritmo para el mismo texto que en el ejemplo anterior, *migraine nausea aspirin*:

Figura 5.18: Ejemplo de resultado 2

6. Verificación y evaluación

6.1 Verificación

6.1.1. Pruebas unitarias para el Crawler

Para comprobar que el funcionamiento del software es correcto, se ha usado PyUnit para testear las funciones de la aplicación.

Para comprobar que el Crawler recoge correctamente la información de patientslikeme.com, se comprobará, de una página web de una enfermedad o tratamiento al azar, que la información se ha recogido como debería y que el contenido de los ficheros resultantes es correcto. Para ello se crearán directorios de prueba, y se comprobará si la información creada en estos ficheros es como debería ser.

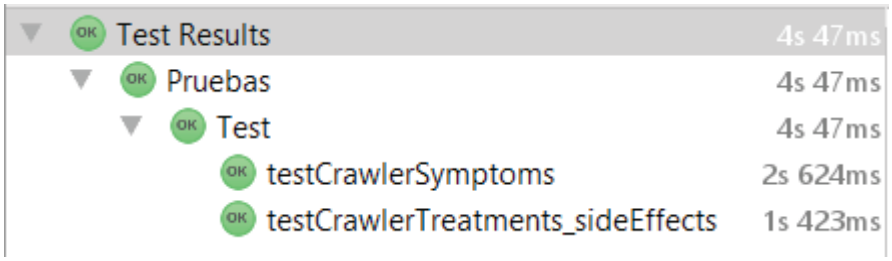


Figura 6.1: Verificación del prototipo 1: crawler

6.1.2. Pruebas unitarias para la creación de la ontología

Para comprobar que el funcionamiento de la creación de la ontología es correcto, se comprobará que las funciones (una por fichero de información) que recogen información por línea de fichero, funcionan correctamente, usando líneas de estos ficheros elegidas al azar.

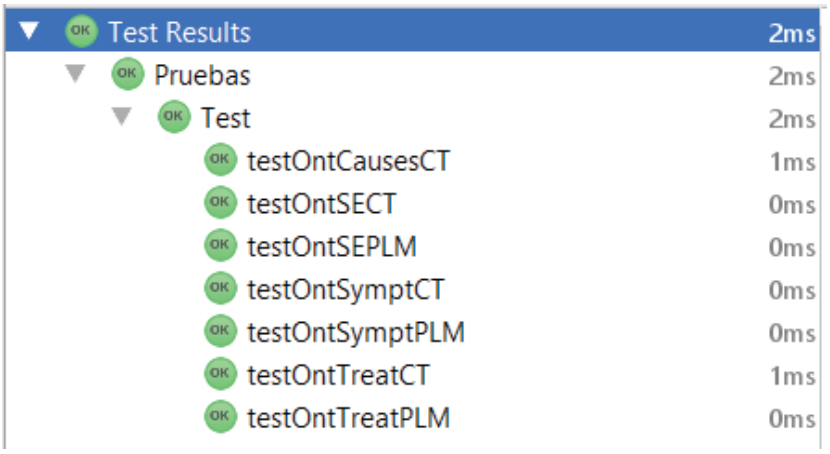


Figura 6.2: Verificación del prototipo2: ontología

6.1.3. Pruebas unitarias para el detector e información

Para asegurar que el detector funciona correctamente, se comprobará que la ejecución de todos sus métodos es como se esperase.

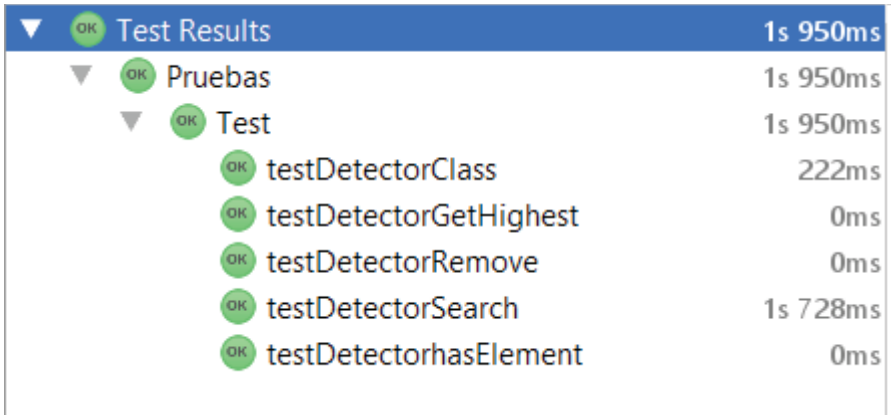


Figura 6.3:Verificación del prototipo 3: Detector

En cuanto a la parte que muestra la información, como únicamente escribe texto en un fichero, se ha testeado que lo que devuelve existe, y no esté vacío. Para comprobar que la información que muestra tiene sentido, se mirará al ejecutarlo y leer el contenido.

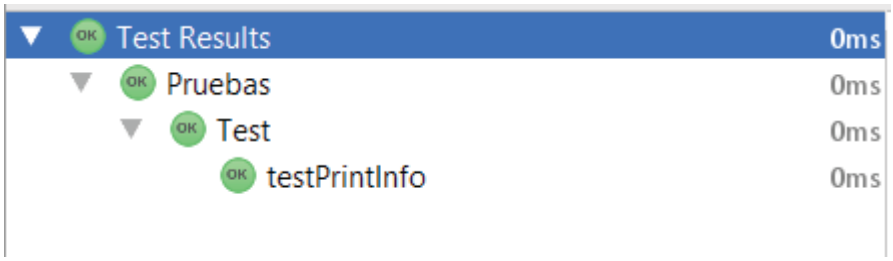


Figura 6.4: Verificación del prototipo 4: información

Para el funcionamiento de las consultas SPARQL, no se han hecho pruebas unitarias, ya que se ha comprobado que están correctamente escritas usando la interfaz de usuario de consultas SPARQL de Apache Jena Fuseki con nuestra ontología cargada, como se puede apreciar en el ejemplo:

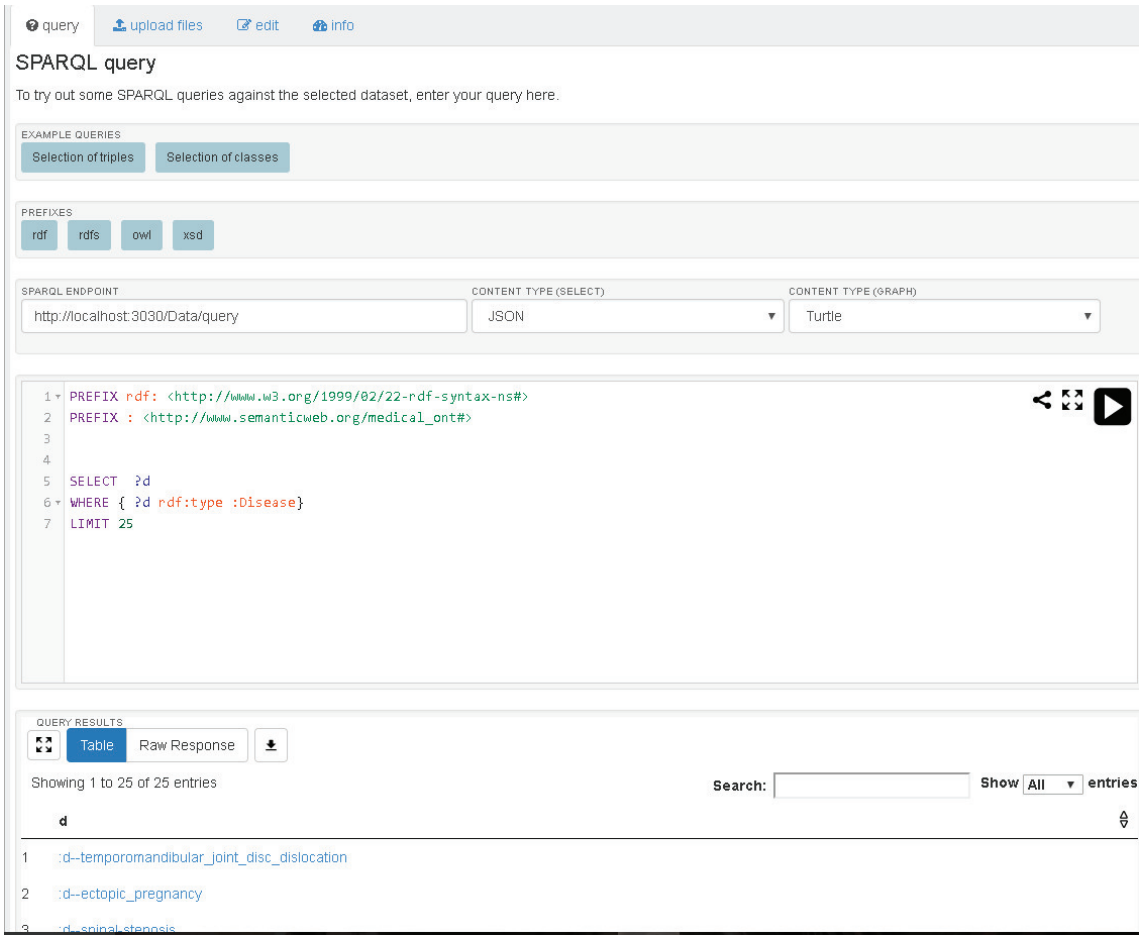


Figura 6.5: Ejemplo consulta SPARQL en Apache Jena Fuseki

6.1 Análisis de los resultados

A la hora de evaluar este proyecto, hay que tener en cuenta las fuentes de los datos. Lo primero de todo, al tratarse de páginas webs donde los usuarios pueden reportar lo que quieran, sin ser expertos, los resultados podrían no ser 100% fiables.

En cuanto a curetogether.com, como los usuarios pueden añadir cualquier síntoma, causa, tratamiento… es posible que algunos de los datos no tengan sentido, ya que la gente que usa estas páginas no tienen que ser especializados en medicina. Han aparecido datos sin ningún tipo de sentido como *surgery* o *car accident* como síntomas.

Para comprobar la cobertura del detector, una vez finalizada la ontología, se siguieron los siguientes pasos:

1- Coger 40 textos del foro de enfermedades de curetogether al azar
2- Etiquetado manual por parte de Carlos y Aitziber
3- Etiquetado automático por parte de la herramienta implementada
4- Comparación de resultados y obtención de figuras de mérito.

Lo primero de todo, cabe mencionar que la tarea de etiquetado no ha sido nada fácil por diversos motivos. Obviando la falta de amplios conocimientos sobre medicina y el idioma en el que están escritos los textos, han aparecido una serie de problemas a la hora de realizar esta tarea. Se han cogido 40 textos al azar de los foros de enfermedades de curetogether y se han identificado los posibles términos. Ya que cualquiera puede poner un mensaje en este foro, hay mensajes de todo tipo, tanto spam, como textos mal escritos, cosas que no tienen sentido, etc.

Antes de continuar, para facilitar la interpretación de las etiquetas, se muestra el índice utilizado a la hora de realizar esta tarea:

Figura 6.6: Índice del etiquetado

Sin embargo, la tarea de los anotadores y su comparación posterior ha ayudado a mejorar el etiquetado, como ha ocurrido en este texto, donde hubo un acuerdo completo en lo referente a las etiquetas:

In college, I learned that Bipolar disorder included manic states that could last for weeks. Because of that, I did not even consider that I may have had anything like it for over 20 years! I thought I was only suffering from anxiety and depression - However, no anti-depressants or anxiety medications helped me. It was only when a therapist recently said I had bipolar tendencies, and put me on lamictal, that I saw a significant improvement. I now know that I have bipolar.

A continuación, se muestran los problemas que han surgido a lo largo del etiquetado, explicados a través de una serie de ejemplos:

1. **Etiquetar la sustancia como tratamiento o el elemento que lo contiene o del de donde se obtiene o ambas.**

A lo largo del etiquetado aparecieron varios términos que englobaban el nombre oficial del tratamiento y del elemento lo contenía o del de donde se obtenía. Ante esta situación, hubo que decidir si optar por seleccionar como tratamiento ambos términos o solo uno. Uno de los textos en los que encontramos este problema fue el siguiente:

Studies show that antioxidants (selenium, N-acetylcysteine etc.), EPA (from fish oil), Vitamin D3 and some minerals (chromium, zinc) help (but require a few months of daily usage to see full results). Sleeping probably helps too (no light at night or even in the evening) due to melatonin (which is a powerful antioxidant also), might also require exposure to bright light

during daytime (to suppress daytime melatonin production, keep in mind that artificial lights are usually very low lumens compared to sunlight, like 10-fold difference).

En este caso podemos ver un tratamiento llamado *EPA* y el elemento del que se obtiene, *fish oil.* En este caso, los anotadores no llegaron a un acuerdo claro de si estiquetar solo *EPA* o ambos.

2. Diferenciar una enfermedad de una causa.

Dentro de los textos, muchas veces ha sido complicado distinguir una enfermedad como tal de una causa que provocaba ciertas consecuencias. Dependía, principalmente, del contexto en el que se encontrase escrito y de cómo se interpretase. Esto es lo que ocurrió en el siguiente ejemplo que se muestra:

Psoriasis get worse when eating chocolate. After 2 weeks on the ocean with sun-bathing psoriasis is gone for 5 month. Maybe I don't have staphylococcus aureus infection in addition to psoriasis Reason: On Sept 2012, I had a car accident with an open fracture; consequently surgery, followed by 6 weeks of extensive antibiotics. After 4 weeks my psoriasis was gone. 3 months after ending antibiotics psoriasis was back.

La dificultad en este texto apareció al encontrar el término *open fracture.* Una fractura como tal se pude considerar una enfermedad, pero también podría ser etiquetada como causa si se relacionase con *staphylococcus aureus infection* o con el hecho de que se le hayan dado al usuario *antibiotics.* Finalmente, se decidió etiquetar *open fracture* como enfermedad.

3. Diferenciar un efecto adverso de un síntoma.

Otra de las dificultades encontradas en este tarea fue clasificar ciertos términos que, según el contexto que exponía el usuario, podrían ser síntomas o efectos adversos provocados por otro término. En el siguiente texto de ejemplo, es lo que se encontró:

I have been on lipitor, and question some memory loss, where I go to say a word and I can't think of a simple word. Very conserting. Am wondering if anyone else had noticed their memory loss. My sister is also on Lipitor and she too has noticed the same affect. Just wondering.

Como se puede observar en el texto aparece el término *memory loss* un par de veces. *Memory loss* podría ser considerado tanto como síntoma, como efecto adverso. Teniendo en cuenta el contexto y la comparación de los diferentes anotadores, se llegó a la conclusión de que era un efecto adverso del tratamiento *Lipitor*.

4. Diferenciar un síntoma de una enfermedad.

Por otro lado, otro de los problemas que han dificultado la realización del etiquetado ha sido diferenciar algunos términos como síntoma o como enfermedad, ya que no estaban del todo claros, por lo que había conflicto. Es esto lo que ocurrió al enfrentarse al siguiente texto:

Mom has small scabs on back, shoulders and shins. They peel away to form hollow lesions. In less than 1 week, most of them are "drying up". What did we take? 1 teaspoon cod liver oil, 1 capsule Vitamin k2 and 1-2 gelcaps of vitamin d3. Nothing, and I mean NOTHING else worked.

En este caso, se encuentran dos términos problemáticos: *scabs* y *hollow lesions*. Ambos términos pueden llegar a considerarse como síntomas de una enfermedad que no ha sido explicitada en el texto por el usuario. Sin embargo, también podrían llegar a ser consideradas como enfermedades tratadas con los tratamientos que se especifican en el texto posteriormente. Tras el debate de los anotadores, se llegó a la conclusión de considerar ambos término como enfermedades.

5. La aparición de palabras mal escritas dentro de los textos.

Como ya se ha dicho, en el foro puede escribir cualquier persona por lo que era evidente que apareciese este problema: la presencia de términos mal redactados, con faltas de ortografía, etc. Tal y como ocurre en el siguiente texto:

I have bad eczema, work in an ER. Think that I am colonized with MRSA. Have been using hibaclens with a good amount of relief but not as much on my face. I think that the hibaclens did help and so do bleach bath.

En este texto se observa el término *hibaclens*. Dado el contexto que deja ver el usuario que ha escrito el comentario, se podría considerar como tratamiento de su enfermedad *eczema*. Sin embargo, al ser una palabra mal escrita, no iba a estar registrada en la ontología. Para confirmar esto, los anotadores se encargaron de buscar el término correcto, que sería *hibiclens* y finalmente se decidió no etiquetar el término erróneo como tratamiento.

6. Seleccionar el tratamiento en su totalidad.

En varias ocasiones, aparecían términos de más de una palabra que podían llegar a considerarse tratamientos. Sin embargo, no fue fácil decidir si etiquetar el conjunto de términos o considerar que era información irrelevante y marcar uno de ellos. Esto es lo que ocurrió, por ejemplo, en el siguiente texto:

A person has to be unbelievably dedicated to achieve a 100% gluten free diet. Every single bite of food that goes in your mouth must be questioned - and you need to know the exact

ingredients to be certain that it is gf. Many restaurants do not know the exact contents of their sauces and dressings, especially if they are corporate chains and the sauces arrive pre-packaged. Things like maltodextrin, soy sauce (used as a marinade at a Mexican restaurant I went to!), some forms of vinegar,

En el comentario anterior, se encuentra el conjunto de términos *gluten free diet.* En este caso, se podría haber etiquetado el conjunto o simplemente el término *diet* como tratamiento de la enfermedad que padeciese el usuario. Finalmente, los anotadores consideraron adecuado etiquetar el conjunto de términos *gluten free diet* como tratamiento, ya que la información que aportaban no era irrelevante.

6.1 Evaluación

Tras etiquetar los textos manual y automáticamente. Se han comparado los resultados de ambos anotadores para calcular el ITA (inter tagger agreement), y los del detector para calcular precision, recall, y f-measure, definidas por las siguientes fórmulas.

$$ITA = \frac{n\ acuerdos}{n\ totales}$$

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$\text{F- Measure} = \frac{Precision * Recall}{Precision + Recall}$$

El valor del ITA obtenido es **0.82**, lo que quiere decir que en general hubo bastante acuerdo, y se anotaron correctamente la mayoría de elementos, pero hubo algunos donde no se llegó a un acuerdo final, como se ha explicado anteriormente.

Para medir la cobertura del detector, se han calculado los elementos que se han detectado correctamente (TP), los que no se han detectado(FN), y los que se han detectado de manera incorrecta(FP) en cada texto y después se ha calculado las figuras de mérito:

	Detectados bien	Detectados mal	No detectados	Precision	Recall	F-Score
Enfermedades	15	1	28	0,9375	0,34883	0,50847
Síntomas	4	29	12	0,12121	0,25	0,16326
Tratamientos	41	8	45	0,836734	0,47674	0,60740
Causas	9	73	6	0,109756	0,6	0,18556
Efectos	2	8	8	0,2	0,2	0,2
TOTAL	71	119	99	**0,373684**	**0,41764**	**0,39444**

Tabla 6.1: Evaluación del detector

Se puede observar, que la precisión de las enfermedades y los tratamientos es bastante alta. Sin embargo, el recall es más bajo. Esto se debe a que hay bastantes enfermedades y tratamientos que han aparecido en los textos y no aparecen en la ontología, y como el detector solo encuentra elementos que pertenecen a esta (y que estén escritos igual), si no están presentes no los va a detectar. Pero los elementos que detecta, la mayoría de veces lo hace correctamente.

En cuanto a los tratamientos, hay que tener en cuenta que algunos pueden ser frases, no solo una o dos palabras, como "*avoid food that cause intestinal gas*". Estos elementos son imposibles de detectar para esta herramienta, a no ser que esa misma frase exacta esté en la ontología, ya que el detector compara los términos tal cual aparecen en esta.

Por otra parte, las causas y síntomas son un auténtico caos, como se puede apreciar, la precisión es muy baja. Esto se debe en parte a que, como la información se ha recogido de páginas en las que los usuarios pueden reportar las causas y síntomas que les apetezca, muchos han añadido cosas sin sentido, y al aparecer éstas en los textos, han sido detectadas. Algunos ejemplos de este tipo de elementos son *surgery*, *car accident*, *back*, *face* como síntomas que no tienen ningún sentido.

Para las causas sin embargo no hay tantos elementos sin sentido, sino que muchas veces detecta algo que es una causa cuando realmente es un síntoma o efecto secundario.

Por último, en cuanto a los efectos secundarios, se puede observar que los resultados tampoco son gran cosa, aunque en los textos han aparecido muy pocos, y en la ontología también es la clase con menos cantidad de elementos.

En resumen, los resultados del detector son bastante mejorables, pero teniendo en cuenta que el funcionamiento del detector es bastante simple, y la información de la ontología contiene datos sin sentido, son mejores de los que se esperaba.

7. Conclusiones y trabajo futuro

7.1 Conclusiones sobre el trabajo

Como opinión personal, este trabajo se centra en un ámbito realmente innovador, como es la recopilación masiva y tratamiento de información médica de Internet. En la actualidad mucha gente recurre a internet para consultar información y dar la suya, y esta información puede ser de gran utilidad para investigaciones.

Por otra parte, este proyecto se ha desarrollado en un lenguaje de programación (Python) y estructura de datos (ontología) que no se conocía, lo que ha supuesto un desafío extra. Aunque a priori parecía que iba a ser más costoso, no me ha resultado muy difícil aprender a programar en Python, se trata de un lenguaje robusto y de fácil comprensión. Lo que ha sido más complejo, es entender cómo funcionan las ontologías, me ha resultado un concepto bastante complejo e interesante, y que se podría llegar a profundizar más si hubiera habido más tiempo.

En cuanto al prototipo 1 y 2, han resultado ser los más complicados y que más tiempo he tenido que invertir para completarlos, ya que había que analizar la información de las páginas web, y aprender a crear una ontología. Además, tratar la información obtenida tampoco ha sido tarea fácil ya que cada fichero de información tenía una estructura totalmente diferente.

Por último, realizar los últimos prototipos en Python ha sido un hándicap extra, ya que no había muchas librerías para crear ontologías y realizar consultas SPARQL en Python, lo que me ha obligado a realizar pasos extras como la creación de un servidor local para acceder a la ontología, escribir yo mismo la sintaxis de la ontología sin usar librerías, o recurrir a un .jar programado personalmente en Java para poder cambiar de formato la ontología.

7.2 Conclusiones personales

El principal reto de este trabajo ha sido el tener que realizarlo de manera individual, ya que, durante la carrera, la mayoría de proyectos que han tenido que ser realizados en las asignaturas, han sido en grupo. Por otra parte, he tenido que aprender a programar en un lenguaje totalmente nuevo para mi, y usar una estructura de datos que no conocía hasta el comienzo del proyecto.

Además, la parte de obtener información de páginas web también comenzó siendo difícil ya que no sabía cómo funcionaba el “crawling”, y la información que tuve que analizar los primeros meses de estas páginas no fue nada fácil, aunque finalmente resulto ser menos de lo que parecía.

Otra dificultad ha sido la falta de tiempo, ya que se ha realizado en un periodo corto, de unos 6 meses, y a la par que cursaba los 48 créditos optativos del cuarto curso, por lo que ha sido necesario invertir muchas horas al día en poco tiempo para poder finalizarlo.

En resumen, estoy contento con el trabajo realizado. He aprendido a trabajar de manera individual y he aprendido a manejar otro lenguaje de programación y otra estructura de datos que no había visto durante la carrera, lo que se traduce como un incremento de todo lo que he aprendido durante mi formación en la UPV/EHU.

7.3. Trabajo futuro

Este proyecto tiene numerosos puntos donde puede ser ampliado o mejorado, como los que se enumeran a continuación:

Lo primero de todo, en cuanto a la ontología, sería de gran interés tratar la información que contiene, ya que, como se ha explicado, hay muchos datos que no tienen mucho sentido, pero que no se han tratado durante este proyecto.

Además, la ontología puede ampliarse, y seguir añadiendo información de diferentes fuentes, consultando el manual de la ontología y siempre y cuando se respete la estructura y el formato en el que están sus elementos. También se puede reagrupar y ordenar los elementos de la ontología, creando nuevas clases y relaciones, por ejemplo, si se desea añadir el código identificativo de cada enfermedad, o agruparlas por tipos (cardiovascular, digestiva…) bastaría con añadir los códigos o tipos como instancias, y relacionarlos con las enfermedades en cuestión con una object property “is-a”.

Por último, también podría mejorarse el funcionamiento del detector, o usar herramientas que ya existan y adaptarlas al proyecto para mejorar los resultados de las consultas.

Anexo 1. Casos de uso extendidos y diagramas de secuencia

Obtener información de la web

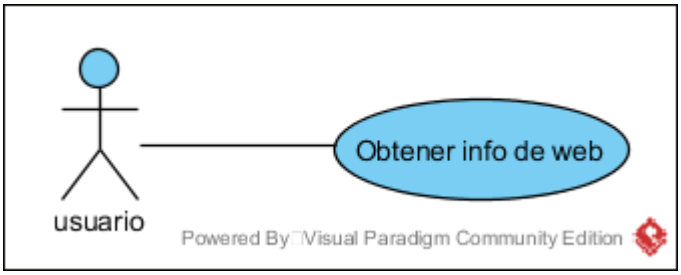


Figura 8.1 Caso de uso extendido: Obtener información de la web

Nombre: Obtener información de la web
Descripción: Cuando el usuario pulsa el botón “Obtener información de la web” se crean los ficheros con la información disponible en patientslikeme sobre enfermedades, síntomas, tratamientos y efectos secundarios.
Actores: Usuario
Precondiciones: Que la página web funcione, y que la estructura de los archivos HTML de las páginas no cambie y coincida con la implementada para obtener la información
Postcondiciones: Se crean 3 ficheros: symptoms_ plm.csv, treatments_ plm.csv y side-effects_plm.csv con la información de estos.

Tabla 8.1 Caso de uso extendido: Obtener información de la web

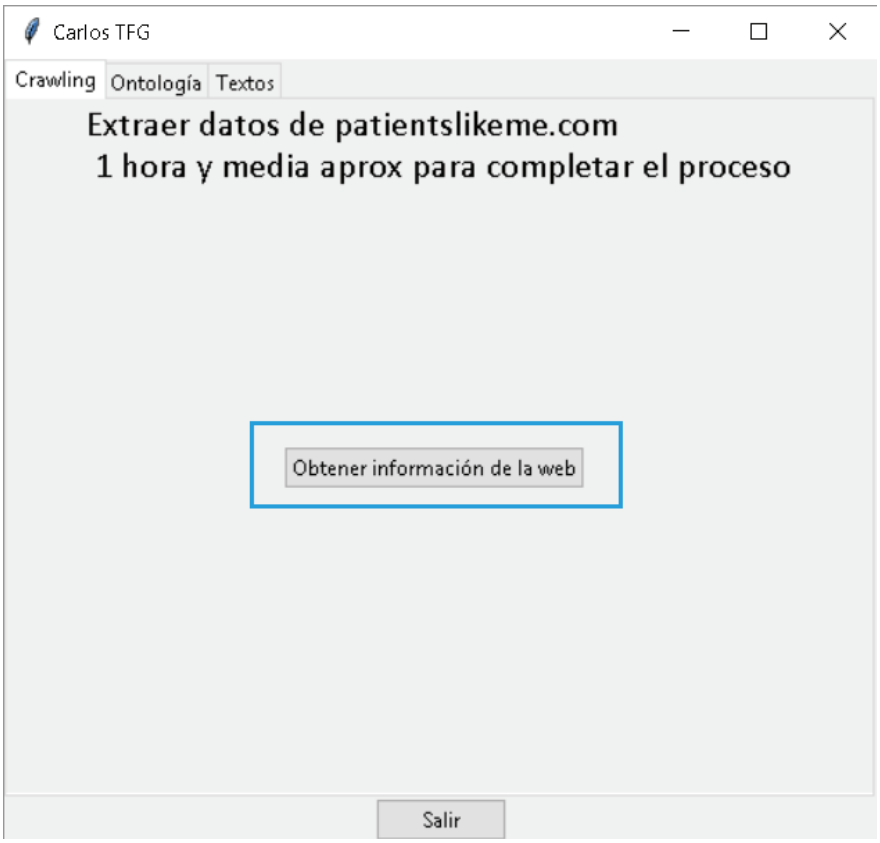


Figura 8.2: Interfaz gráfica: Obtener información de la web

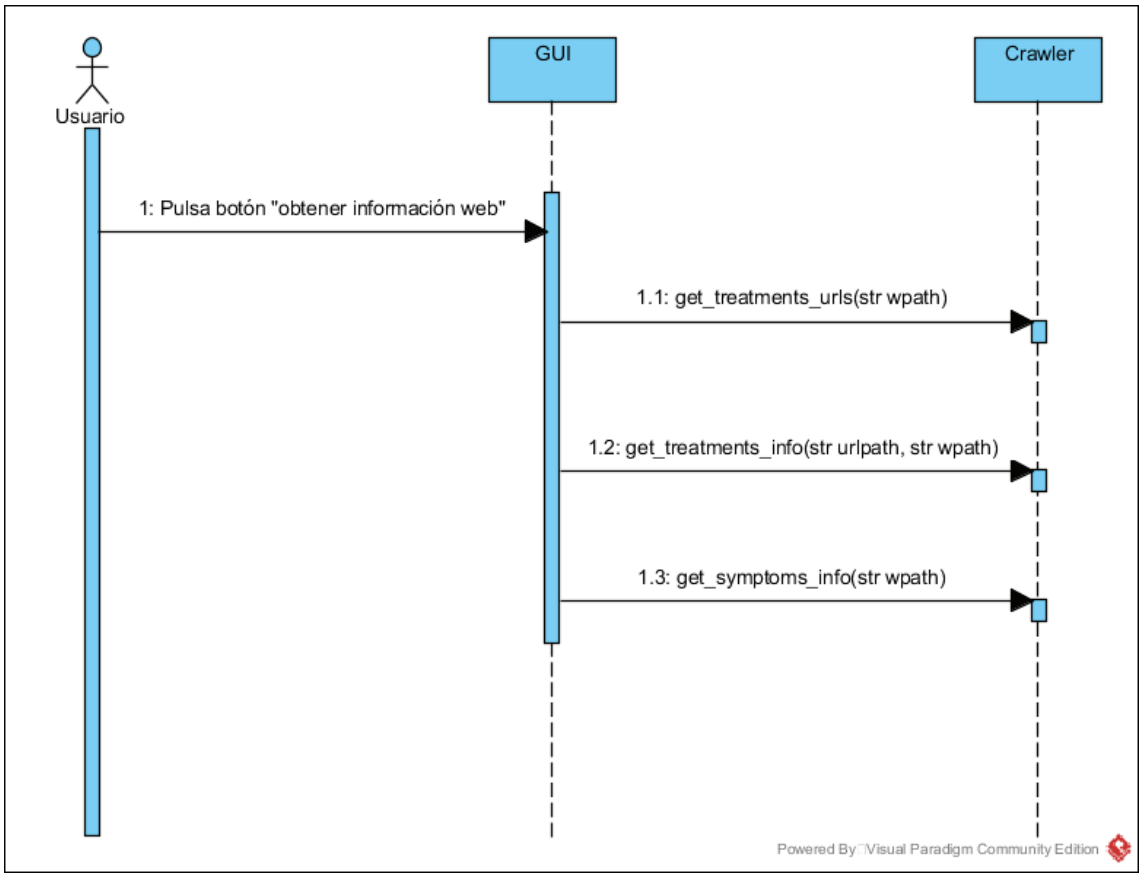


Figura 8.3: Diagrama de secuencia: Obtener información de la web

Volcar información a la ontología

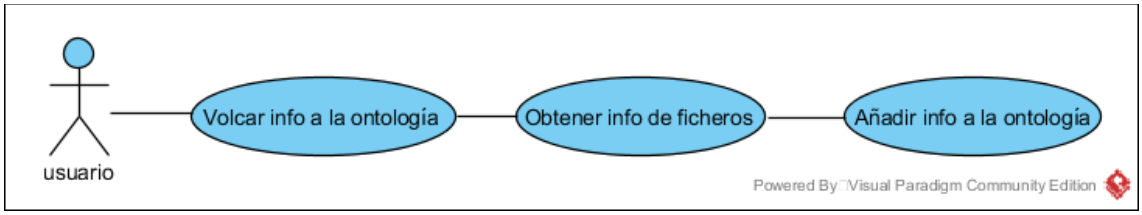


Figura 8.4: Caso de uso extendido: Volcar info a la ontología

Nombre: Volcar información a la ontología
Descripción: Cuando el usuario pulsa el botón "Crear ontología", primero se recogerá la información de los ficheros con los datos de las webs, y posteriormente se volcarán estos datos en un fichero en formato OWL/XML el cual será el fichero de la ontología. Se llamará data.owl
Actores: Usuario
Precondiciones: Que existan los ficheros con la información de las webs, y que el contenido de éstos coincida con las expresiones regulares implementadas.
Postcondiciones: Se creará una ontología en un fichero .owl con la información de las webs en un directorio especificado.

Tabla 8.2: Caso de uso extendido: Volcar info a la ontología

Figura 8.5: Interfaz gráfica: Volcar info a la ontología

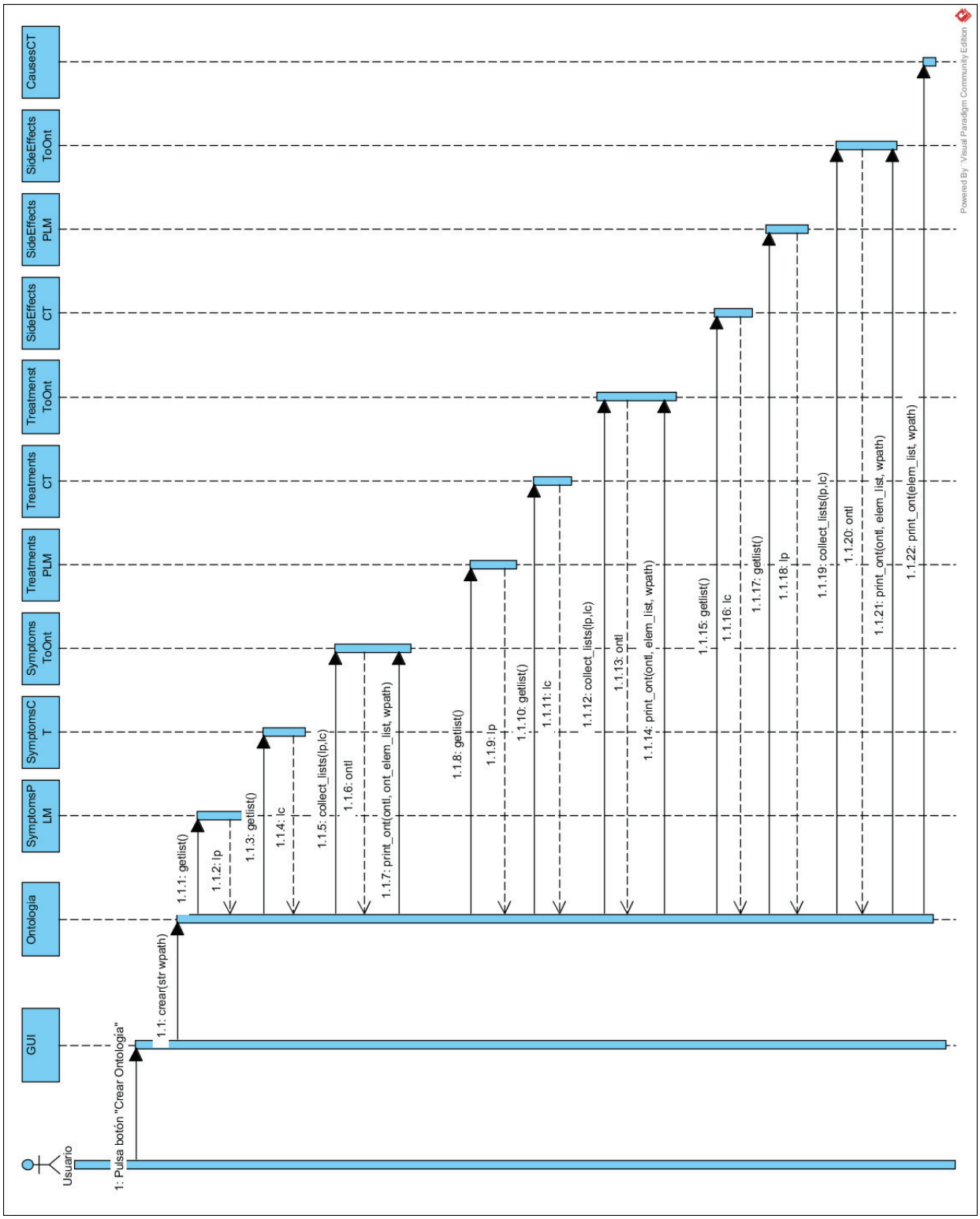


Figura 8.6: Diagrama de secuencia: Volcar info a la ontología

Dar información de un texto

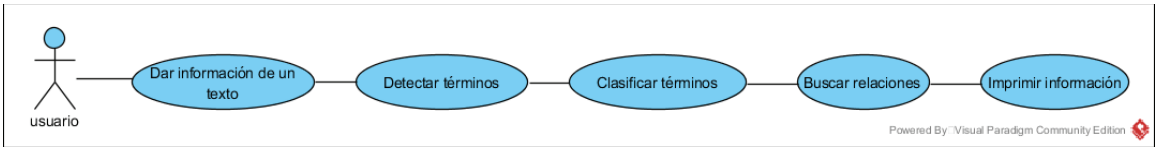


Figura 8.7: Caso de uso extendido: Dar información de un texto

Nombre: Dar información de un texto
Descripción: Cuando el usuario pulse el botón "Commit", a partir del texto que se escriba, primero se detectarán los términos que se haya reconocido de la ontología, se clasificarán por tipos, se buscarán relaciones entre ellos y se imprimirá esa información. Además, si el usuario lo desea, puede obtener además la información de elementos relevantes para los que se hayan encontrado en el texto (síntomas de una enfermedad, efectos de un tratamiento etc.).
Actores: Usuario
Precondiciones: Que la ontología esté disponible en un servidor local.
Postcondiciones: Se mostrará por pantalla los resultados, y si se desea guardar, se creará un .txt con la información.

Tabla 8.3: Caso de uso extendido: Dar información de un texto

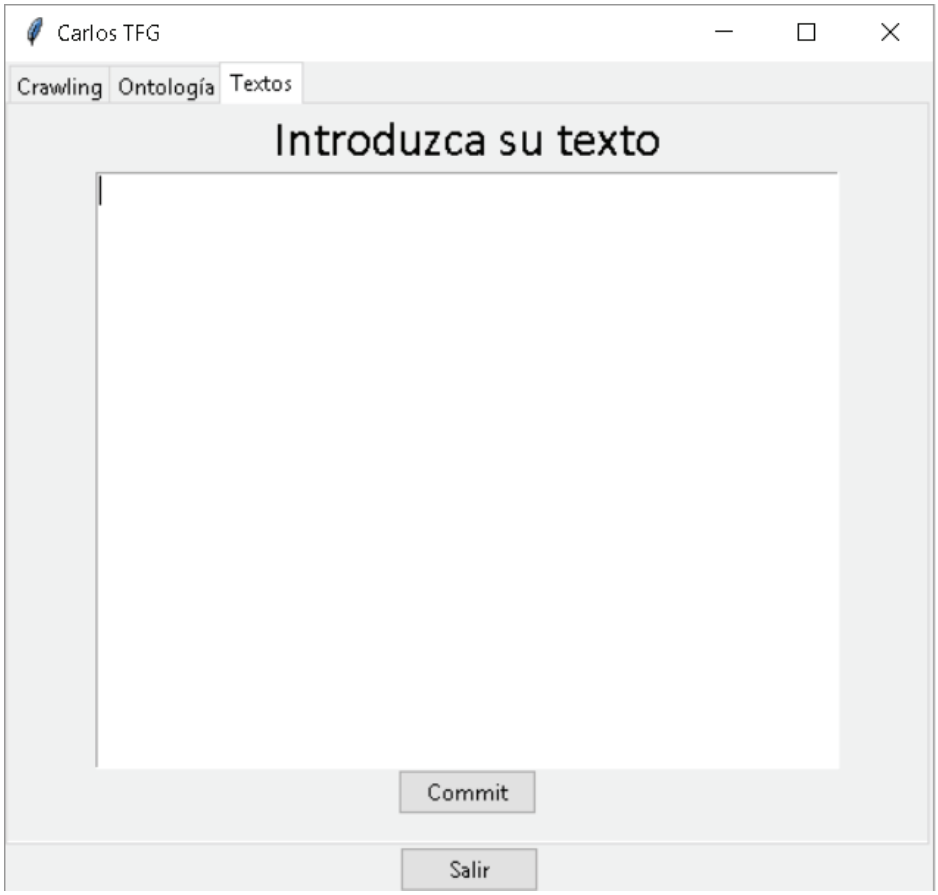


Figura 8.8: Caso de uso extendido: Dar información de un texto

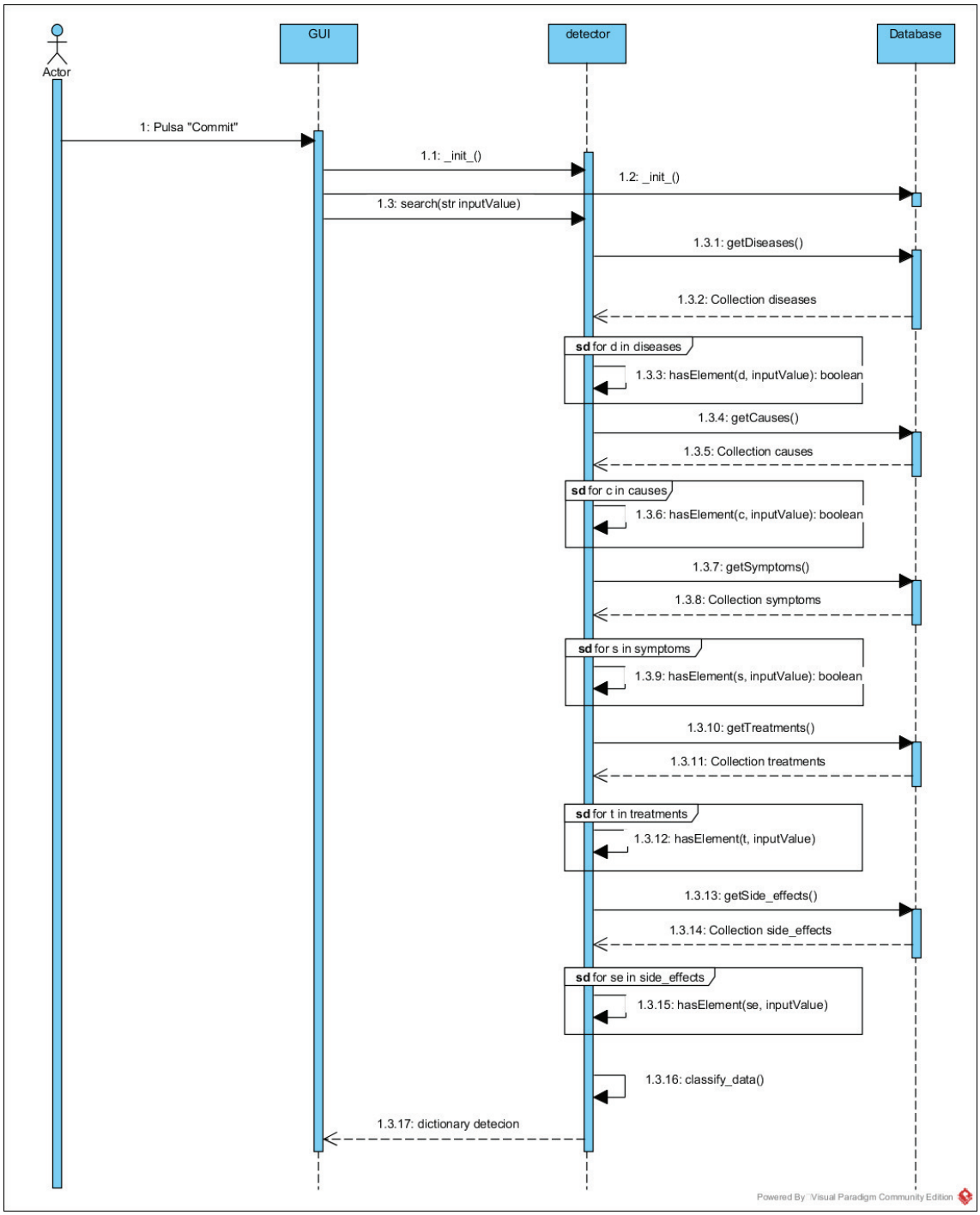


Figura 8.9: Caso de uso extendido: Dar información de un texto (Detectar términos)

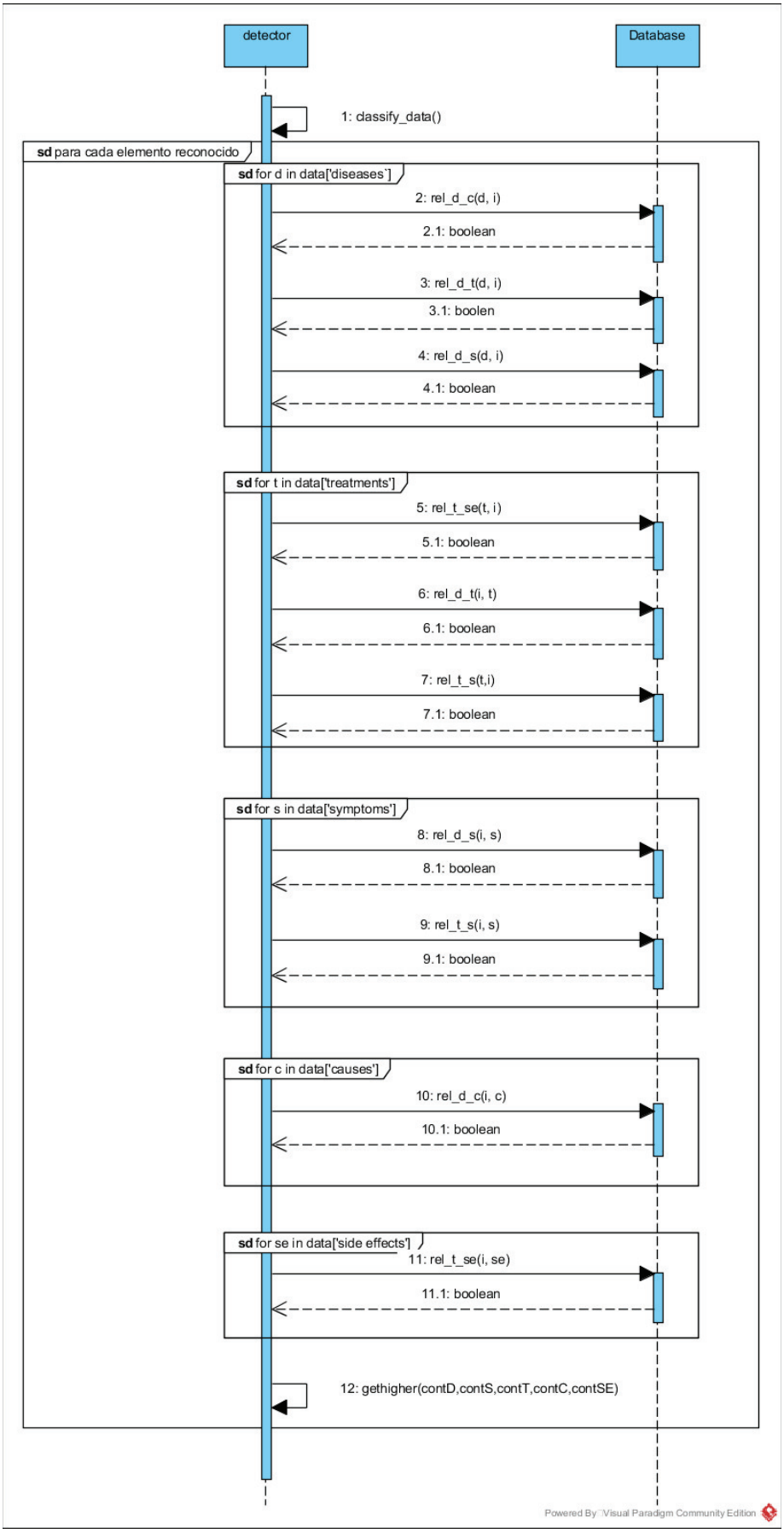


Figura 8.10: Caso de uso extendido: Dar información de un texto (Clasificar términos)

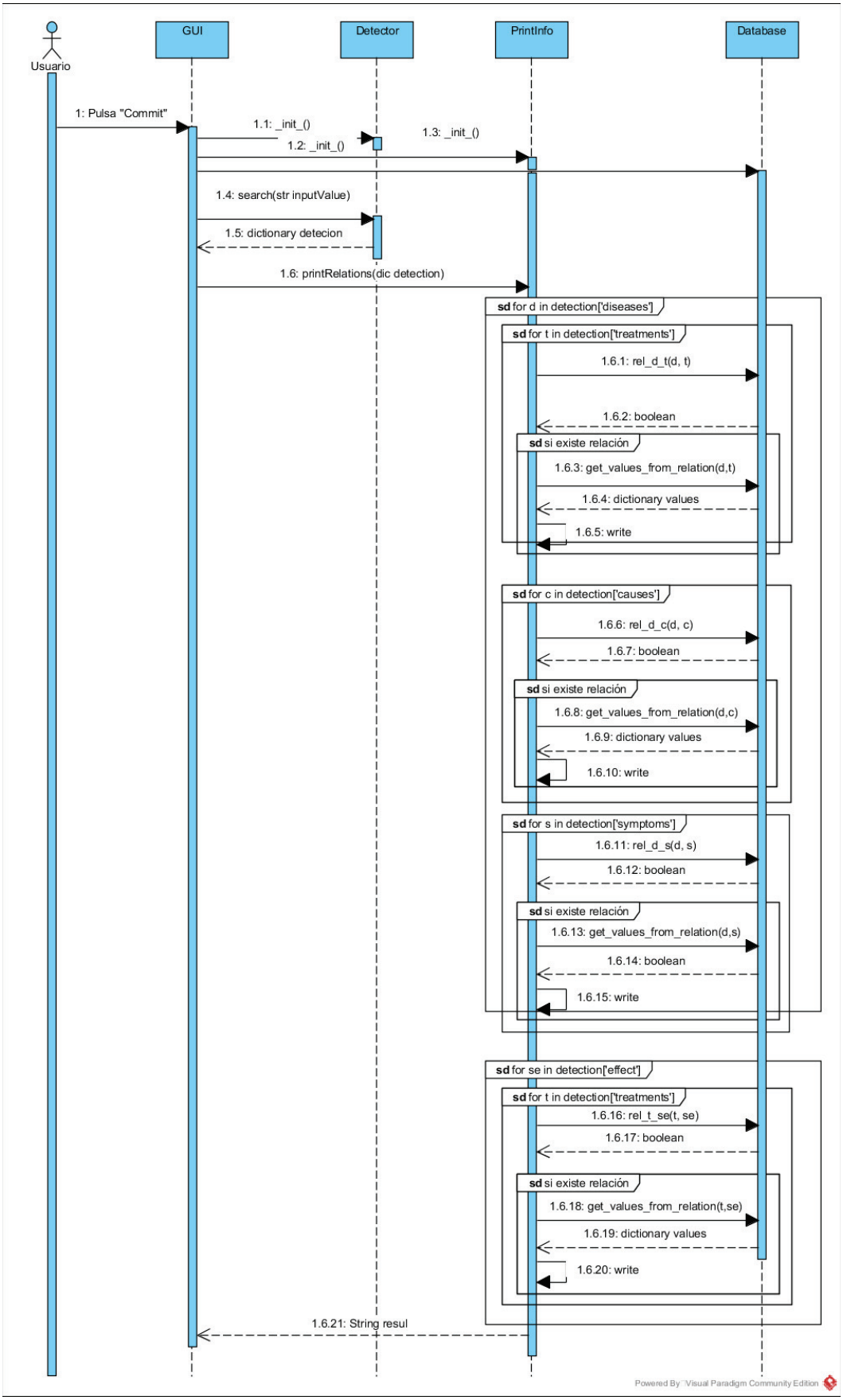


Figura 8.11: Caso de uso extendido: Dar información de un texto (Imprimir información)

Anexo 2. Manual de ontología

A continuación, se va a explicar cómo se podría ampliar la información existente en la ontología.

Cargar la ontología

Con la herramienta **Protégé 4.3** se puede cargar la ontología creada por la aplicación sin ningún problema, y ver todas las instancias, clases y relaciones. Para ello basta con ir a File-Open (Ctrl +O) y seleccionar el fichero de la ontología, que se llamará data.owl.

Desde código, para cargar la ontología para hacer consultas, se necesitará la librería **SPARQLWrapper**. Además, es necesario cargar la ontología previamente en un servidor local para que SPARQLWrapper funcione, para ello también será necesario **Apache Jena Fuseki**. Para iniciar el servidor y cargar la ontología, bastará con ejecutar el archivo .jar de jena fuseki. Usando el siguiente comando:

```
'cd java -jar path_del_archivo.jar && fuseki-server.bat --file "
                path_del_fichero_de_ontología" /Data'
```

Con esto hemos creado un dataset llamado "Data" con la información de la ontología. Se puede comprobar que se ha cargado correctamente, y realizar SPARQL queries desde la interfaz de este servidor, siendo por defecto localhost:3030

Una vez la ontología está cargada en el servidor, ya podemos usar la librería SPARQLWrapper para cargarla en nuestro código Python, para ello bastará con la siguiente línea de código (El nombre del dataset creado anteriormente tiene que coincidir con el que se va a llamar en el código siguiente para que funcione correctamente el programa):

```
import SPARQLWrapper
self.sparql = SPARQLWrapper("http://localhost:3030/Data/query")
```

Para añadir información a la ontología, **no es necesario cargarla de este modo**, esto sólo es necesario para hacer consultas a la ontología. Para añadir información sólo es necesario **cargar el fichero de la ontología** simplemente usando la función *open* en **modo append** como en el siguiente ejemplo:

```
ontfile = open(path_del_fichero_de_ontología, 'a')
```

Además, antes de añadir información al fichero, es necesario **eliminar la última línea** que cierra la ontología (</Ontology>) antes de añadir nada. Cuando se haya acabado, es necesario volver a añadir esta línea. En los apartados siguientes se especifica cual es la sintaxis para añadir cada tipo de información en **formato OWL/XML** necesarios. Para ello, **habrá que transformarla previamente** a este formato, ya que la aplicación lo transforma en **formato RDF/XML** para poder usar la librería SPARQLWrapper**.** Para ello, se puede usar Protégé muy fácilmente: cargamos el archivo de nuestro proyecto "data.owl" y le damos a File-"Save as" y allí nos dará a elegir en que formato se desea guardar, le damos a OWL/XML y ya tendríamos la ontología en este formato. En el apartado 5.2.2 de la memoria del proyecto, se explica cómo es el formato de los elementos de la ontología.

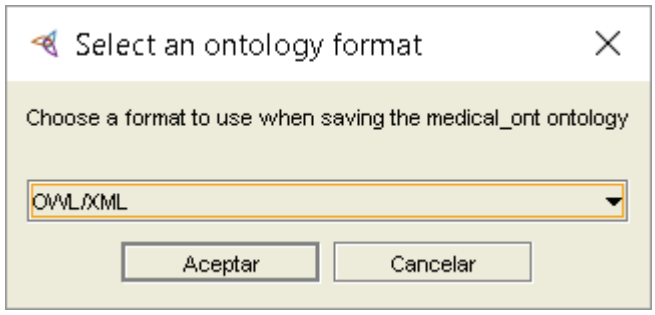


Figura 9.1 Selección formato ontología

También hay un jar llamado **rdf2owl** que realiza esta misma función si se desea hacer desde Python. El comando que habría que ejecutar para transformar el formato del fichero de la ontología es el siguiente:

```
os.system("java -jar "+path_del_jar+" " + '"' +path_del_fichero_ont+
'"')
```

Y el fichero ya estaría en formato OWL/XML

Este paso **no es necesario** si se desea escribir en formato RDF/XML, ya que es en el que se encuentra por defecto, pero como en el proyecto se ha usado OWL/XML, es el que se procede a explicar.

Añadir una nueva clase

Para añadir una nueva clase a la ontología con Protégé, bastará con ir a la pestaña "Entities", luego a la pestaña "Classes", y pulsar el botón "Add subclass" para añadir una subclase a una clase previamente creada, o "Add sibling class" para añadir una clase al mismo nivel que otra ya creada.

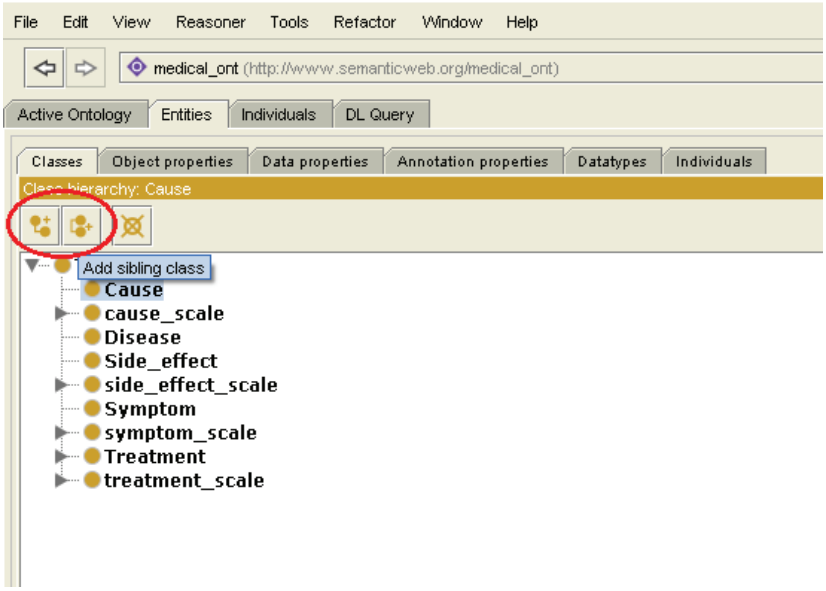


Figura 9.2: Añadir nueva clase Protégé

El fragmento de código OWL/XML que habría que añadir al fichero para añadir una nueva clase es el siguiente:

```
<Declaration>
        <Class IRI="#NOMBRE_CLASE"/>
</Declaration>
```

Si se desea crear una subclase, de una ya existente, el código es el siguiente:

```
<SubClassOf>
        <Class IRI="#NOMBRE_CLASE_HIJA "/>
        <Class IRI="#NOMBRE_CLASE_PADRE"/>
</SubClassOf>
```

Eliminar una clase

Para añadir eliminar una clase a la ontología con Protégé, es necesario seleccionar la clase que se desea eliminar y pulsar el botón "Delete selected classes". Hay que tener en cuenta que todas las instancias que pertenezcan a la clase que se desea borrar, no serán eliminadas, sino que solamente dejarán de pertenecer a la clase en cuestión.

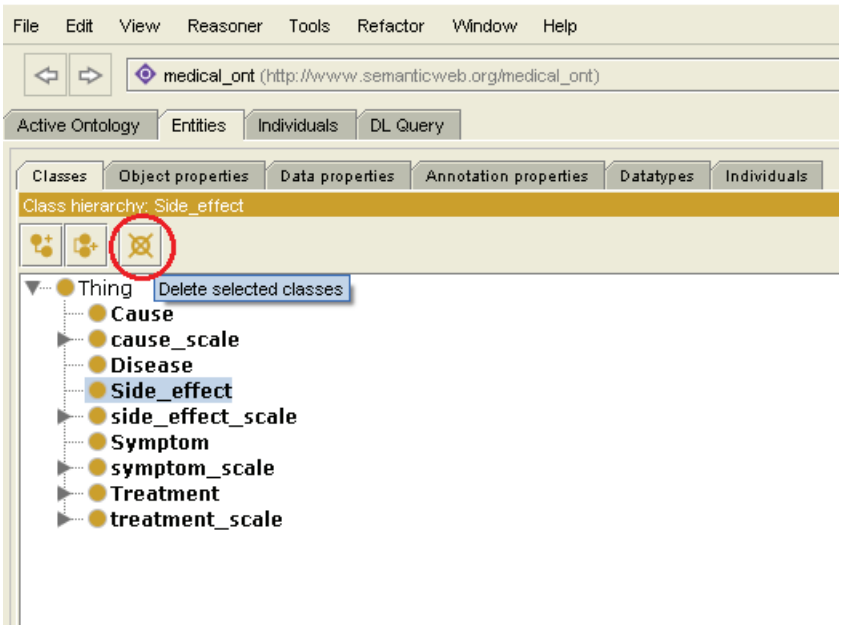


Figura 9.3 Eliminar una clase Protégé

Añadir una relación

Para añadir una nueva relación (Object property), en la pestaña "Entities" hay que seleccionar la pestaña "Object properties" y pulsar el botón "Add sub property" para añadir una subproperty a

una clase previamente creada, o “Add sibling property” para añadir una property al mismo nivel que otra ya creada.

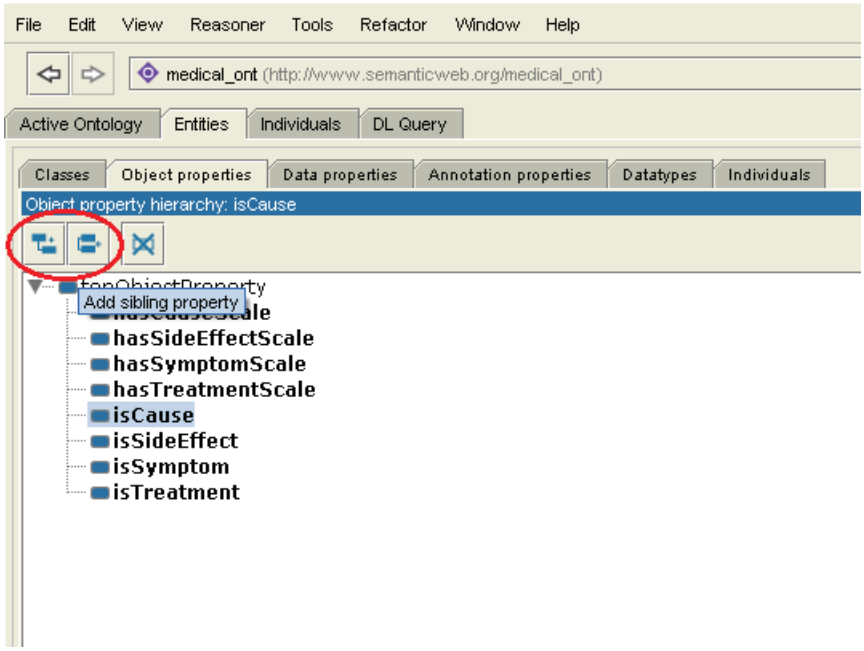


Figura 9.4 Añadir Object Porperty Protégé

El segundo paso, es especificar el dominio y el rango de la relación. Para ello, estando seleccionada la property en cuestión, en la parte derecha de la pantalla, en la sección “Description”, pulsamos el botón “Add” de los apartados “Domains” y “Ranges” se eligen la clase que se desee.

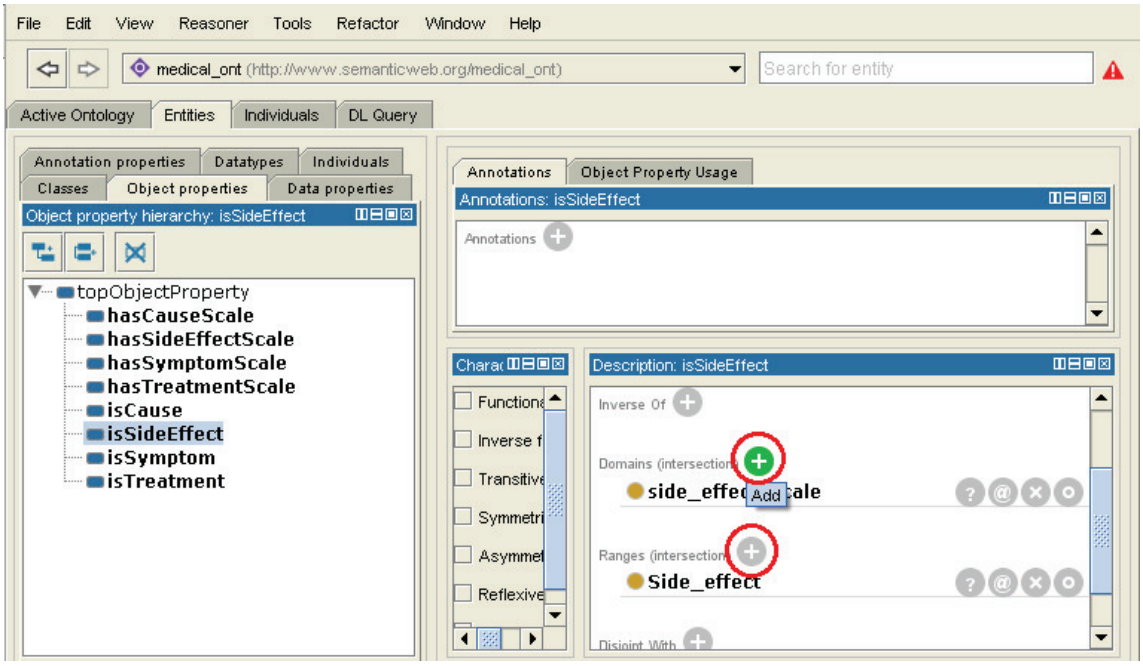


Figura 9.5 Editar Object Property Protégé

El fragmento de código OWL/XML que habría que añadir al fichero para crear una Object property es el siguiente:

Para añadir la Object property:

```
<Declaration>
        <ObjectProperty IRI="#NOMBRE_RELACIÓN"/>
</Declaration>
```

Para especificar dominio y rango de una relación:

```
<ObjectPropertyDomain>
        <ObjectProperty IRI="#NOMBRE_RELACIÓN"/>
        <Class IRI="#NOMBRE_CLASE_DOMINO"/>
</ObjectPropertyDomain>

<ObjectPropertyRange>
        <ObjectProperty IRI="#NOMBRE_RELACIÓN"/>
        <Class IRI="#NOMBRE_CLASE_RANGO"/>
</ObjectPropertyRange>
```

Eliminar una relación

Para añadir eliminar una relación de la ontología con Protégé, al igual que para las clases, es necesario seleccionar la relación que se desea eliminar y pulsar el botón “Delete selected property”.

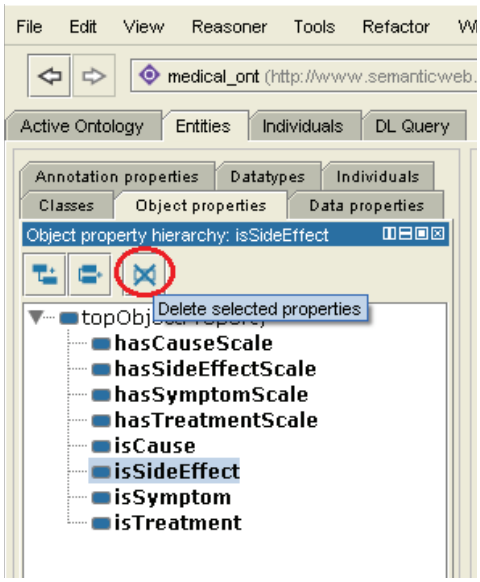


Figura 9.6: Eliminar Object Property Protégé

Crear una “data property”

Para añadir una nueva data property, en la pestaña “Entities” hay que seleccionar la pestaña “Data properties” y pulsar el botón “Add sub property” para añadir una subproperty a una clase previamente creada, o “Add sibling property” para añadir una data property al mismo nivel que otra ya creada.

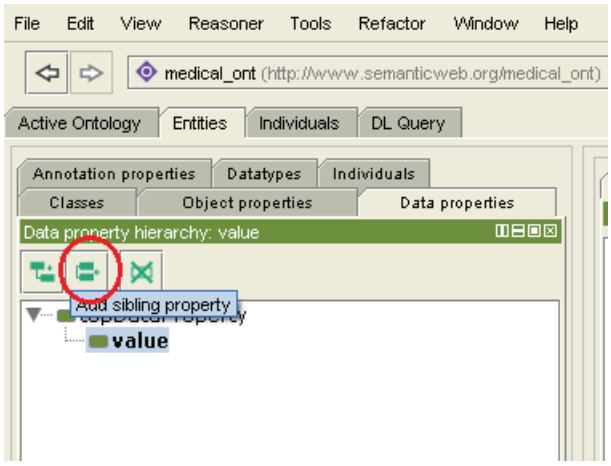


Figura 9.7 Añadir Data Property Protégé

Posteriormente, para indicar de que tipo va a ser el atributo o property, lo seleccionamos, nos vamos a la parte derecha de la pantalla, en la sección “Description”, pulsamos el botón “Add” del apartado “Ranges” y especificamos el tipo.

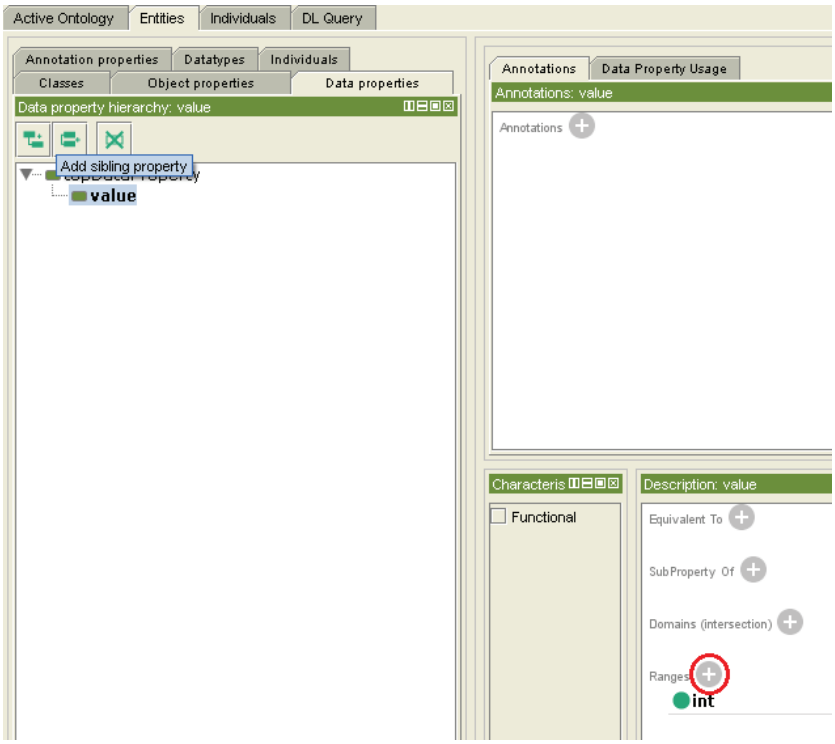


Figura 9.8: Editar Data Property Protégé

Crear, eliminar y asignar elementos a una instancia

Para añadir una nueva instancia (Individual) a la ontología, con Protégé, en la pestaña "Entities" hay que seleccionar la pestaña "Individual" y pulsar el botón "Add individual". Para eliminar una instancia, basta con pulsar el botón de al lado "Delete individual". Hay que tener en cuenta que, al eliminar una instancia, **se eliminan también todas las Object properties** en las que estuviera involucrada.

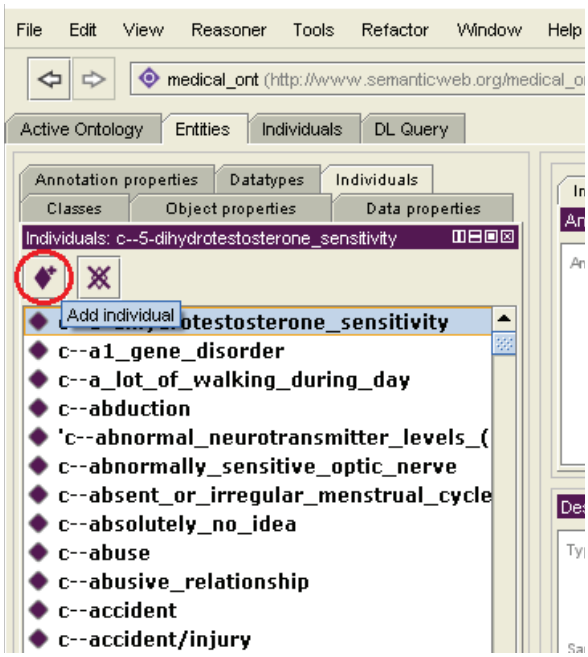


Figura 9.9: Añadir instancia Protégé

Para especificar la **clase de una instancia**, seleccionamos el "invididual", nos vamos a la parte derecha, en la sección "Description", pulsamos el botón "Add" del apartado "Types" y elegimos la clase que queramos.

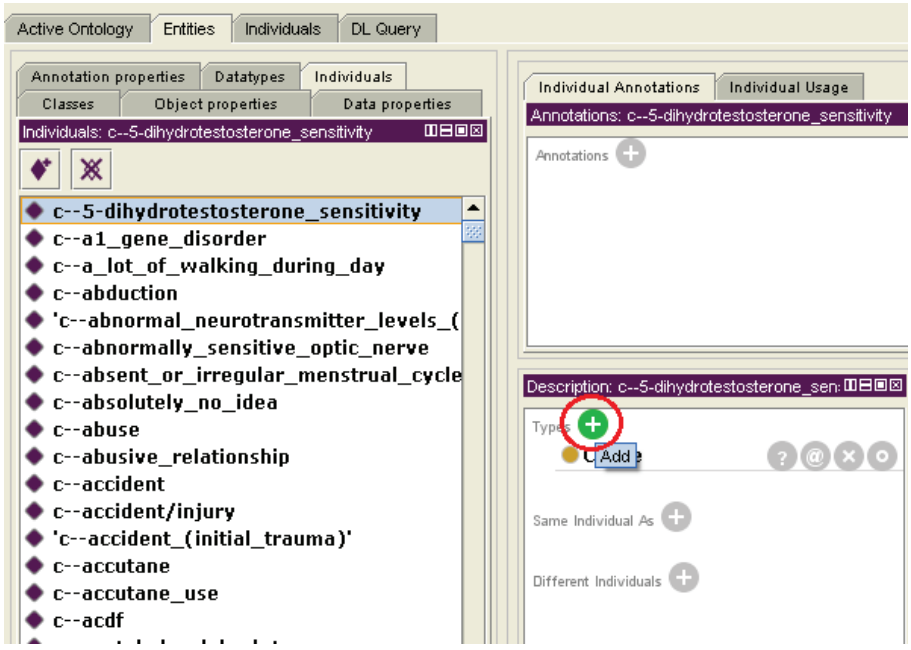


Figura 9.10: Editar instancia Protégé

Por otro lado, para añadir una **relación con otra instancia (Object property)**, nos vamos a la sección "Property assertions", y en el apartado "Object property assertions" pulsamos el botón "Add".

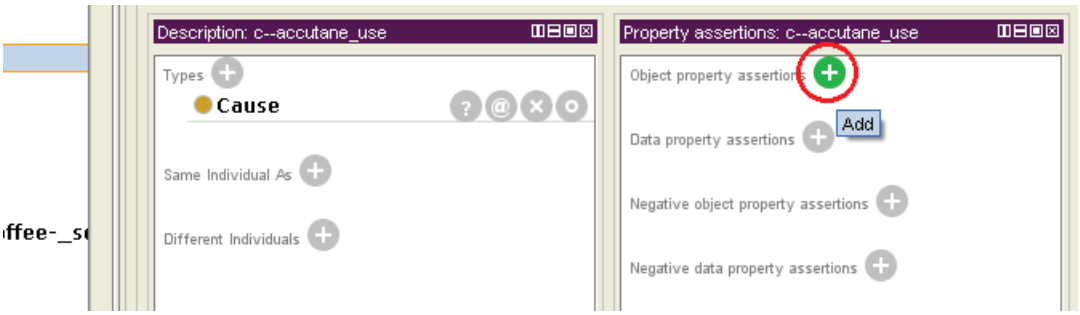


Figura 9.11: Editar instancia Protégé

Tras esto, nos saldrá otra ventana en la que tendremos que especificar qué Object property queremos añadir, y con qué instancia va a estar relacionada.

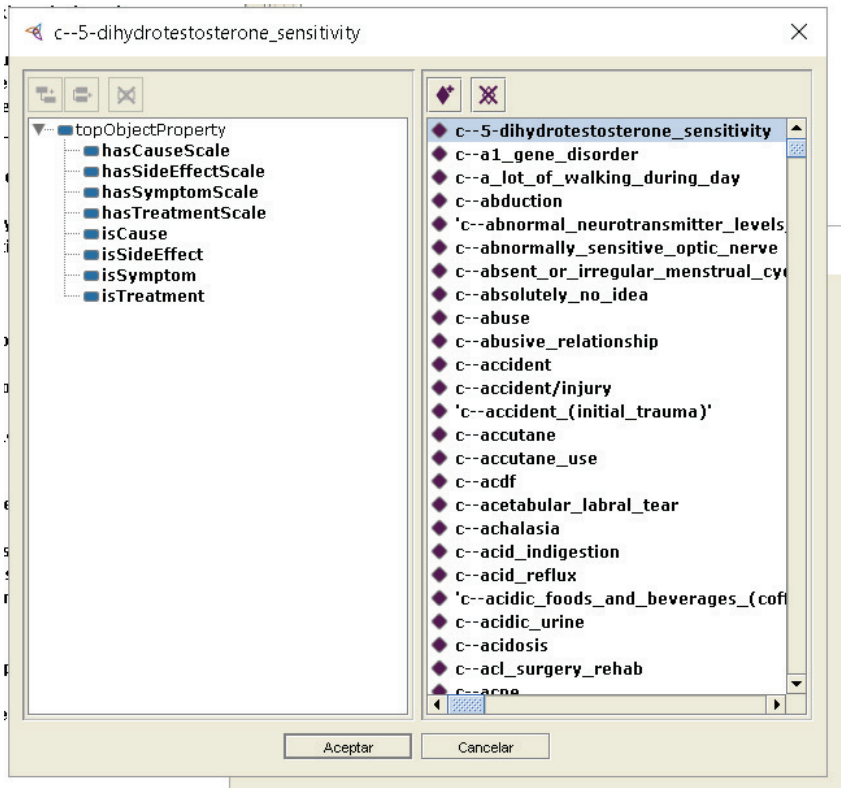


Figura 9.12: Editar instancia Protégé

Hay que tener en cuenta, que la instancia desde la que se añade la relación (la que hemos seleccionada) será el **dominio** y la seleccionada en la segunda pantalla será el **rango.**

Por último, para añadir un **atributo a una instancia**, o, dicho de otra forma, una relación con una **data property**, seleccionamos la instancia y nos vamos nuevamente a la sección "Property" y pulsamos el botón "Add" del apartado "Data property assertion".

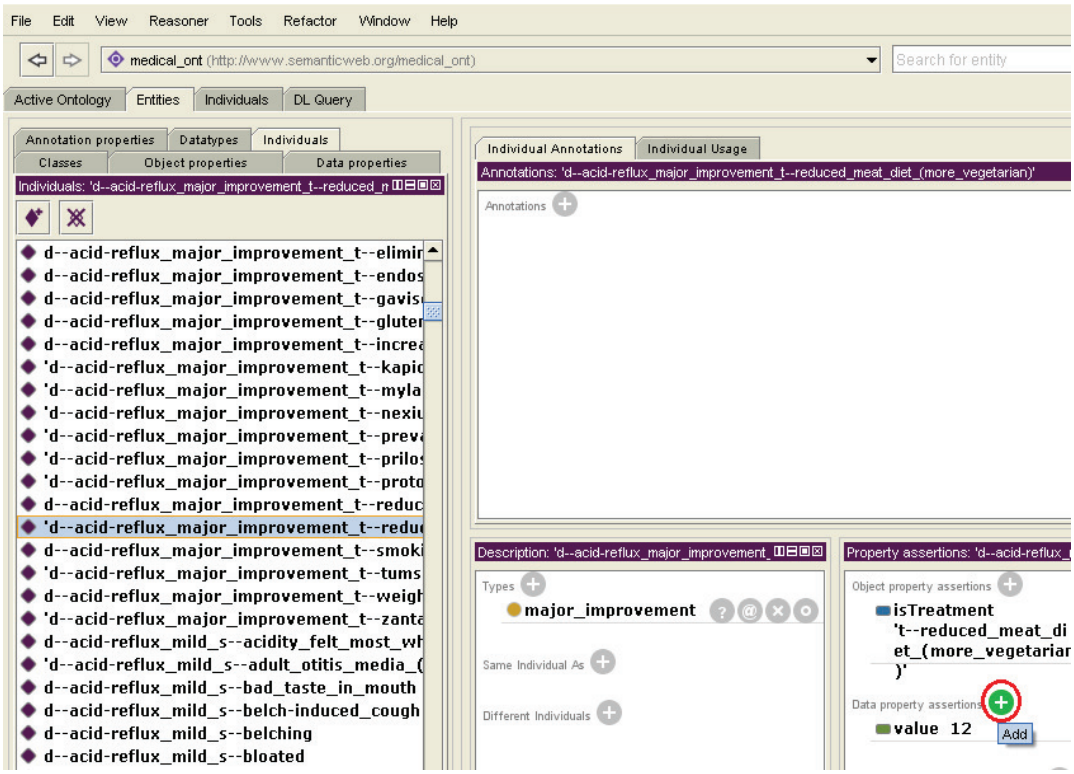


Figura 9.13: Editar instancia Protégé

Tras esto, nos aparecerá otra pantalla, donde tendremos que seleccionar la data property que queramos, y el valor que le vamos a dar.

Para realizar todo esto con código, habría que añadir al fichero de la ontología lo siguiente:

Para crear la instancia:

```
<Declaration>
        <NamedIndividual IRI="#NOMBRE_INSTANCIA"/>
</Declaration>
```

Para asignarle una clase:

```
<ClassAssertion>
         <Class IRI="#NOMBRE_CLASE_ASIGNADA"/>
         <NamedIndividual IRI="#NOMBRE_INSTANCIA"/>
</ClassAssertion>
```

Para crear una relación entre dos instancias (Object property):

```
<ObjectPropertyAssertion>
        <ObjectProperty IRI="#NOMBRE_OBJECT_PROPERTY"/>
        <NamedIndividual IRI="#NOMBRE_INSTANCIA_DOMINO"/>
        <NamedIndividual IRI="#NOMBRE_INSTANCIA_RANGO"/>
</ObjectPropertyAssertion>
```

Para añadir un atributo a una instancia (Data property):

```
<DataPropertyAssertion>
   <DataProperty IRI="#NOMBRE_DATA_PROPERTY"/>
   <NamedIndividual IRI="#NOMBRE_INSTANCIA"/>
   <Literal datatypeIRI="TIPO_DE_LA_DATA_PROPERTY"> VALOR_ATRIBUTO
   </Literal>
</DataPropertyAssertion>
```

Anexo 3. Manual de usuario

Una vez hemos seguido los pasos del manual de instalación, y se ha abierto la interfaz, el funcionamiento de esta es muy simple. Es importante no modificar ningún nombre ni ningún archivo de lugar, ya que el programa podría dejar de funcionar correctamente.

Hay tres secciones:

1. **Crawler**: en esta sección solo habrá un botón, el cual al ser pulsado iniciara la tarea de recogida de información de patientslikeme.com y guardará sus datos en los ficheros del proyecto(infocrawling/*.csv) sobreescribiendo la información que haya en estos. Ya que se van a procesar un número muy alto de páginas, esta tarea puede durar aproximadamente una hora y media o dos, dependiendo de las características del equipo.
2. **Ontología**: en esta segunda sección dispone de un botón, que al pulsarlo se volcará los datos de los ficheros de crawling en un fichero .owl, que será el de la ontología. Se guardará en (ontología/data.owl). Esta tarea dura aproximadamente 5-10 minutos.
3. **Consulta de información**. En esta tercera sección, habrá un text box para escribir lo que el usuario desee, tal y como se muestra a continuación

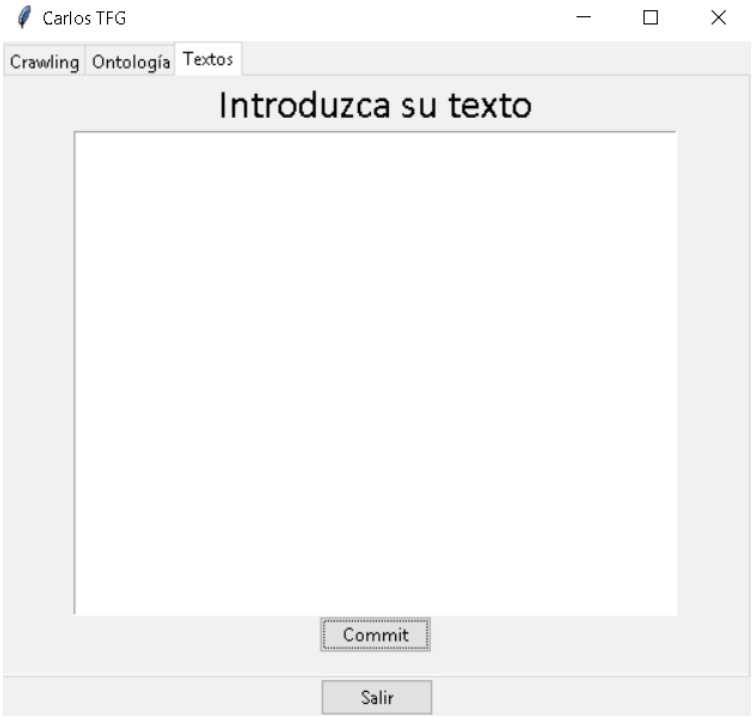


Figura 10.1: Pantalla 3 del sistema

,

Al pulsar el botón Commit, el programa buscará las relaciones entre los elementos que haya detectado de ese texto, y mostrará en otra ventana el resultado al usuario.

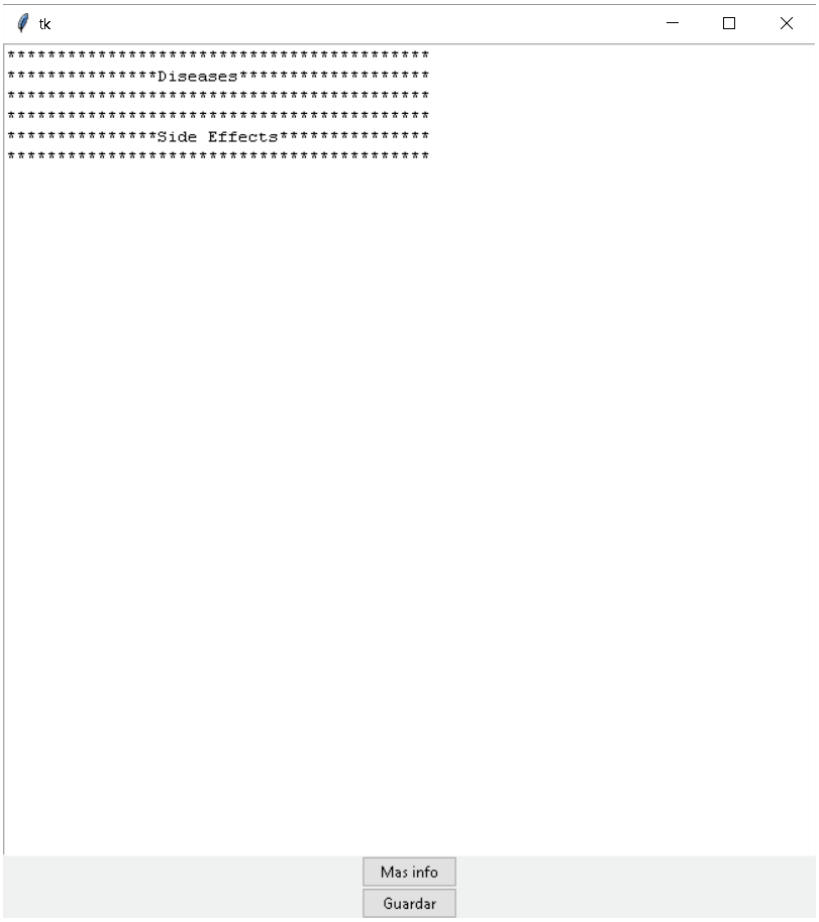


Figura 10.2: Pantalla resultado del texto

El usuario podrá decidir si quiere que se le muestre información extra pulsando el botón "Mas info", que se tratará de elementos relevantes para cada uno detectado, y se mostrarán en una pantalla similar. El usuario podrá guardar los resultados de estos dos resultados si pulsa el botón "Guardar". Una pantalla informativa informará al usuario dónde se han guardado los resultados como la siguiente:

Figura 10.3: Pantalla informativa de guardado

Bibliografía

Apache Sowftware Foundation. (2017). *jena.apache.org*. Obtenido de jena.apache.org: https://jena.apache.org/documentation/fuseki2/index.html

Cal Poly . (2011). *wiki.csc.calpoly.edu*. Obtenido de wiki.csc.calpoly.edu: https://wiki.csc.calpoly.edu/OntologyTutorial/wiki/IntroductionToOntologiesWithProtege

Graciela Barchini, Margarita Álvarez, Susana Herrera y Melina Trejo. (2007). El rol de las ontologías en los SI. *Revista ingeniería informática*.

JetBrains. (2017). *Jetbrains*. Obtenido de Jetbrains: https://www.jetbrains.com/pycharm/

Python Software Foundation. (09 de Abril de 2012). *Regular expression operations*. Obtenido de https://docs.python.org/3.1/library/re.html

Python Software Foundation. (2014). *pypi.python.org*. Obtenido de pypu.python.org: https://pypi.python.org/pypi/SPARQLWrapper/1.6.4

Python Software Foundation. (2017). *pypi.python.org*. Obtenido de pypi.python.org: https://pypi.python.org/pypi/beautifulsoup4

Worl Wide Web Consorcium. (15 de Junio de 2008). *SPARQL Query Language for RDF*. Obtenido de https://www.w3.org/TR/rdf-sparql-query/

World Wide Web Consortium. (12 de Noviembre de 2009). *OWL Web Ontology Language*. Obtenido de https://www.w3.org/TR/owl-features/