POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

TRABAJO DE FIN DE GRADO

Herramienta basada en vistas para la generación automática de anotaciones de fuentes RDF

Autor: Andoni Lloret Carbonell
Director: Miguel García Remesal
Alberto Anguita Sánchez

MADRID, JUNIO DE 2014

Nunca es tarde

ÍNDICE DE CONTENIDO

RESUMEN......vi
ABSTRACT......viii
INTRODUCCIÓN......1
1.1 Planteamiento del problema......1
1.2 Objetivos del trabajo......4
1.2.1 Solución propuesta......4
ESTADO DE LA CUESTIÓN......5
2.1 Algoritmos léxicos......5
2.2 Algoritmos semánticos......7
2.4 Alineamiento de Ontologías......8
2.5 Alineamiento de Vistas......8
TECNOLOGÍAS EMPLEADAS......10
3.1 Lenguajes de programación......10
3.1.1 Java......10
3.1.2 JavaScript......11
3.2 Otros lenguajes......12
3.1.1 HTML......12
3.2.2 RDF......12
3.2.3 OWL......12
3.3 APIs externas......14
3.3.1 MappingAPI2......14
3.3.2 OWLBasicModel2......15
3.3.3 SemanticEquivalence......15
3.3.4 JAWS (Java API for WordNet Searching)......16
3.3.5 JDOM......16
3.4 Software empleado......17
MÉTODOS......19
4.1 Algoritmo para creación de vistas de la base de datos física......19
4.1.1 Caso práctico de recorrido del esquema físico......20
4.2 Algoritmo de alineamiento automático......22
4.2.1. Diagramas de flujo del algoritmo semi-automatico......26
4.3 Construcción de frases en lenguaje natural a partir de vistas RDF......28
ANÁLISIS DEL SISTEMA......30
5.1 Características de la herramienta......30
5.2 Especificación de requisitos del software......33
5.2.1 Introducción......33
5.2.2 Descripción Global......36
5.2.3 Requisitos Específicos......39
5.3 Casos de uso del Sistema......44
5.3.1 Actores......44
5.3.2 Diagrama de Casos de Uso......44

5.3.3 Casos de Uso....45
5.3.4 Diagramas de Secuencia del Sistema....48
5.3.2 Contratos de las operaciones del Sistema....51
DISEÑO E IMPLEMENTACIÓN DE LA HERRAMIENTA....56
6.1 Arquitectura de la herramienta....56
6.2 Interfaz Web....57
6.3 Aplicación en el servidor....58
6.3.1 Clases....58
EXPERIMENTOS Y RESULTADOS....63
7.1 Experimentos....65
7.1.1 Tasa de aciertos sin sinónimos....65
7.1.2 Tasa de aciertos con sinónimos....66
7.1.3 Umbrales Damerau-Levenshtein....67
7.2 Caso Práctico....69
CONCLUSIONES Y LÍNEAS FUTURAS....73
8.1. Conclusiones....73
8.2 Líneas futuras....76
BIBLIOGRAFÍA....78

ÍNDICE DE FIGURAS

Figura 1: Fórmula Jaro-Winkler....6
Figura 2: Triple....8
Figura 3: Path....9
Figura 4: Vista e Internal Bound....9
Figura 5: XML generado por el MappingAPI2....15
Figura 6: XML del progreso almacenado....17
Figura 7: esquema RDF representado mediante un grafo dirigido....21
Figura 8: De 3 clases físicas salen 6 triples en la parte conceptual....23
Figura 9: Crecimiento (n-1)*n de las triples que se pueden generar....24
Figura 10: Generación de una estructura de similitud....26
Figura 11: Generación de una vista conceptual con las triples de menor peso....27
Figura 12: Vista RDF de un solo path con dos triples....29
Figura 13: Pantalla principal de la herramienta....30
Figura 14: Detalle del recuadro de acciones para un término....31
Figura 15: Detalle del recuadro de acciones para una relación....32
Figura 16: Diagrama de Casos de Uso del Sistema....45
Figura 17: Diagrama de secuencia de Gestión de Sinónimos....48
Figura 18: Diagrama de secuencia de Acciones sobre Clases y Relaciones....49
Figura 19: Diagrama de secuencia de Acciones sobre Vistas....50
Figura 20: Diagrama de secuencia de Gestionar progreso y resultado....50
Figura 21: Arquitectura de la herramienta....56
Figura 22: Clase ViewContainer....58
Figura 23: Clase SimilarityStructure....59
Figura 24: Clase PrintText....60
Figura 25: Clase ActionsClass....60
Figura 26: Clase ActionsProperties....60
Figura 27: Clase Buttons....61
Figura 28: Clase SerializeBox....61
Figura 29: Clase ConceptualViewGenerator....61
Figura 30: Representación gráfica de la base de datos ClinicalDB1....64
Figura 31: Representación gráfica de la base de datos ClinicalDB2....64
Figura 32: Punto de partida en la base de datos ClinicalDB2....69
Figura 33: Punto de partida de ClinicalDB2....70
Figura 34: El usuario añade la palabra “patient” como sinónimo de “pt”....70
Figura 35: Tras aceptar la vista se genera la siguiente....71
Figura 36: El diccionario contiene "neoplasm" como sinónimo de "tumor"....71
Figura 37: Se especifica la vista a partir de la clase biobank_sample....72
Figura 38: Al generalizar la vista se parte desde biobank_sample....72
Figura 39: Al llegar al 100% del progreso el trabajo habrá terminado....72

ÍNDICE DE TABLAS

Tabla 1: Acrónimos propios a la especificación de requisitos..34
Tabla 2: Definiciones propias a la especificación de requisitos.....................................35
Tabla 3: Caso de Uso: Gestionar sinónimos de Clases..46
Tabla 4: Caso de Uso: Acciones sobre Clases y Relaciones...47
Tabla 5: Caso de Uso: Acciones sobre Vistas..48
Tabla 6: Caso de Uso: Gestionar progreso y resultado..48
Tabla 7: Contrato de la operación: addSynonym(clasName,synonym)..........................51
Tabla 8: Contrato de la operación: removeSynonym(clasName,synonym)....................51
Tabla 9: Contrato de la operación: discardClass(className)..52
Tabla 10: Contrato de la operación: discardProperty(propertyName)............................52
Tabla 11: Contrato de la operación: generalizeClass(className)..................................52
Tabla 12: Contrato de la operación: specifyClass(className)......................................52
Tabla 13: Contrato de la operación: setClassAsIdentifier(className)..........................53
Tabla 14: Contrato de la operación: setClassRestriction(className, operand, operator)
....53
Tabla 15: Contrato de la operación: acceptView(view)...53
Tabla 16: Contrato de la operación: discardView(view)..53
Tabla 17: Contrato de la operación: generalizeView(view)...54
Tabla 18: Contrato de la operación: specifyView(view)..54
Tabla 19: Contrato de la operación: saveProgress(Path)..54
Tabla 20: Contrato de la operación: loadProgress(Path)..54
Tabla 21: Contrato de la operación: generateMapping(Path)...55
Tabla 22: Características de las bases de datos físicas...64
Tabla 23: Análisis de tasa de aciertos sin sinónimos para ClinicalDB1.........................65
Tabla 24: Análisis de tasa de aciertos sin sinónimos para ClinicalDB2.........................65
Tabla 25: Resultados de tasa de aciertos sin sinónimos...66
Tabla 26: Análisis de tasa de aciertos con sinónimos para ClinicalDB1........................67
Tabla 27: Análisis de tasa de aciertos con sinónimos para ClinicalDB2........................67
Tabla 28: Resultados de tasa de aciertos con sinónimos..67
Tabla 29: Cantidad de similitudes en función del umbral Damerau-Levenshtein...........68
Tabla 30: Mediciones de tiempo para la generación de la estructura de similitudes.......68

RESUMEN

Durante los últimos años, el imparable crecimiento de fuentes de datos biomédicas, propiciado por el desarrollo de técnicas de generación de datos masivos (principalmente en el campo de la genómica) y la expansión de tecnologías para la comunicación y compartición de información ha propiciado que la investigación biomédica haya pasado a basarse de forma casi exclusiva en el análisis distribuido de información y en la búsqueda de relaciones entre diferentes fuentes de datos. Esto resulta una tarea compleja debido a la heterogeneidad entre las fuentes de datos empleadas (ya sea por el uso de diferentes formatos, tecnologías, o modelizaciones de dominios). Existen trabajos que tienen como objetivo la homogeneización de estas con el fin de conseguir que la información se muestre de forma integrada, como si fuera una única base de datos. Sin embargo no existe ningún trabajo que automatice de forma completa este proceso de integración semántica.

Existen dos enfoques principales para dar solución al problema de integración de fuentes heterogéneas de datos: Centralizado y Distribuido. Ambos enfoques requieren de una traducción de datos de un modelo a otro. Para realizar esta tarea se emplean formalizaciones de las relaciones semánticas entre los modelos subyacentes y el modelo central. Estas formalizaciones se denominan comúnmente anotaciones. Las anotaciones de bases de datos, en el contexto de la integración semántica de la información, consisten en definir relaciones entre términos de igual significado, para posibilitar la traducción automática de la información. Dependiendo del problema en el que se esté trabajando, estas relaciones serán entre conceptos individuales o entre conjuntos enteros de conceptos (vistas). El trabajo aquí expuesto se centra en estas últimas.

El proyecto europeo p-medicine (FP7-ICT-2009-270089) se basa en el enfoque centralizado y hace uso de anotaciones basadas en vistas y cuyas bases de datos están modeladas en RDF. Los datos extraídos de las diferentes fuentes son traducidos e

integrados en un *Data Warehouse*. Dentro de la plataforma de p-medicine, el Grupo de Informática Biomédica (GIB) de la Universidad Politécnica de Madrid, en el cuál realicé mi trabajo, proporciona una herramienta para la generación de las necesarias anotaciones de las bases de datos RDF. Esta herramienta, denominada Ontology Annotator ofrece la posibilidad de generar de manera manual anotaciones basadas en vistas. Sin embargo, aunque esta herramienta muestra las fuentes de datos a anotar de manera gráfica, la gran mayoría de usuarios encuentran difícil el manejo de la herramienta , y pierden demasiado tiempo en el proceso de anotación. Es por ello que surge la necesidad de desarrollar una herramienta más avanzada, que sea capaz de asistir al usuario en el proceso de anotar bases de datos en p-medicine. El objetivo es automatizar los procesos más complejos de la anotación y presentar de forma natural y entendible la información relativa a las anotaciones de bases de datos RDF. Esta herramienta ha sido denominada *Ontology Annotator Assistant*, y el trabajo aquí expuesto describe el proceso de diseño y desarrollo, así como algunos algoritmos innovadores que han sido creados por el autor del trabajo para su correcto funcionamiento. Esta herramienta ofrece funcionalidades no existentes previamente en ninguna otra herramienta del área de la anotación automática e integración semántica de bases de datos.

ABSTRACT

Over the last years, the unstoppable growth of biomedical data sources, mainly thanks to the development of massive data generation techniques (specially in the genomics field) and the rise of the communication and information sharing technologies, lead to the fact that biomedical research has come to rely almost exclusively on the analysis of distributed information and in finding relationships between different data sources. This is a complex task due to the heterogeneity of the sources used (either by the use of different formats, technologies or domain modeling). There are some research proyects that aim homogenization of these sources in order to retrieve information in an integrated way, as if it were a single database. However there is still now work to automate completely this process of semantic integration.

There are two main approaches with the purpouse of integrating heterogeneous data sources: Centralized and Distributed. Both approches involve making translation from one model to another. To perform this task there is a need of using formalization of the semantic relationships between the underlying models and the main model. These formalizations are also calles annotations. In the context of semantic integration of the information, data base annotations consist on defining relations between concepts or words with the same meaning, so the automatic translation can be performed. Depending on the task, the ralationships can be between individuals or between whole sets of concepts (views). This paper focuses on the latter.

The European project p-medicine (FP7-ICT-2009-270089) is based on the centralized approach. It uses view based annotations and RDF modeled databases. The data retireved from different data sources is translated and joined into a *Data Warehouse*. Within the p-medicine platform, the Biomedical Informatics Group (GIB) of the Polytechnic University of Madrid, in which I worked, provides a software to create annotations for the RDF sources.

. This tool, called Ontology Annotator, is used to create annotations manually. However, although *Ontology Annotator* displays the data sources graphically, most of the users find it difficult to use this software, thus they spend too much time to complete the task. For this reason there is a need to develop a more advanced tool, which would be able to help the user in the task of annotating p-medicine databases. The aim is automating the most complex processes of the annotation and display the information clearly and easy understanding. This software is called Ontology Annotater Assistant and this book describes the process of design and development of it. as well as some innovative algorithms that were designed by the author of the work. This tool provides features that no other software in the field of automatic annotation can provide.

CAPÍTULO 1
INTRODUCCIÓN

1.1 Planteamiento del problema

A lo largo de los últimos años ha habido un gran crecimiento de bases de datos biomédicas. La investigación biomédica moderna se basa en la búsqueda de relaciones entre datos de diferentes fuentes de datos, en gran medida heterogéneas entre sí. El análisis de los datos requiere la previa homogeneización de los mismos, lo cual supone un gran gasto de tiempo y recursos. Existen métodos para, en mayor o menor medida, automatizar los procesos de homogeneización de datos heterogéneos, pero aún en la actualidad este es un problema no resuelto completamente. Las soluciones actuales se basan en ocultar las heterogeneidades a los usuarios y mostrar la información de manera integrada, como si fuera de una única base de datos. Sin embargo, a día de hoy ninguna solución permite automatizar de forma completa este proceso de integración semántica.

Existen dos enfoques[1] principales para dar solución a la integración de fuentes heterogéneas de datos: Centralizado y Distribuido. A continuación se describen ambos enfoques.

Centralizado

En este modelo toda la información obtenida de diversas fuentes se traduce y almacena un *Data Warehouse (DW).* Toda la información almacenada en el *DW* sigue un modelo común de datos. Las consultas se hacen directamente a este almacén de datos.

Distribuido

En vez de realizar un volcado a un almacén central, los datos se mantienen en todo momento en sus fuentes originales. Estas bases de datos son mantenidas y desarrolladas por sus respectivos propietarios. Se define un modelo virtual sobre el que los usuarios realizan las consultas. Dichas consultas son traducidas de forma dinámica a los modelos de las bases de datos subyacentes, y los resultados de cada una traducidos a su vez al modelo virtual e integrados para responder a la consulta del usuario.

El enfoque centralizado tiene la ventaja de ser más eficiente a la hora de resolver consultas de usuarios, pues toda la información está disponible en un mismo almacén. La principal desventaja es la necesidad de actualizar de manera periódica el almacén central para evitar que el usuario reciba como respuesta a sus consultas datos desactualizados. El enfoque federado resuelve el problema de los datos desactualizados, pero el proceso de resolver cada consulta es más costoso. En ambos enfoques se ha de realizar una traducción de datos de un modelo a otro (de los modelos de las bases de datos integradas al modelo del *DW*, en el caso del enfoque centralizado, o del modelo virtual, en el caso del modelo distribuido). Para posibilitar que esta traducción de datos se realice de forma automática, se emplean formalizaciones de las relaciones semánticas entre los modelos subyacentes y el modelo central. Dichas formalizaciones, comúnmente denominadas "anotaciones" son relaciones de términos de igual significado entre dos modelos de datos. Dependiendo del tipo de anotación empleado, los términos descritos podrán ser conceptos individuales o vistas completas del modelo en cuestión.

Las anotaciones entre dos bases de datos pueden incluir relaciones semánticas y sintácticas. Estas últimas se dan en los casos en que entre ambas fuentes de datos existen diferencias de formato o tecnología, como puede ser en el caso de bases de datos creadas bajo paradigmas diferentes (relacional, XML…). Para evitar la complejidad que esto conlleva, en los últimos años se ha impuesto el paradigma RDF como modelo de

descripción de información en el área de la biomedicina.

RDF [2] (Resource Description Framework) es una especificación de W3C [3] (World Wide Web Consortium) que permite modelar datos mediante una estructura XML. Su versatilidad permite transformar bases de datos de cualquier otro modelo a RDF, evitando que las anotaciones de datos tengan que especificar heterogeneidades a nivel sintáctico. El trabajo descrito en este libro asume anotaciones entre fuentes de datos basadas en RDF.

Para hacer frente a los diferentes problemas presentados por la heterogeneidad de las bases de datos existen numerosos proyectos de investigación entre los que se encuentra p-medicine (FP7-ICT-2009-270089), el cual engloba a 20 grupos de distintos países. p-medicine se basa en el enfoque centralizado, hace uso de anotaciones basadas en vistas, las bases de datos están modeladas en RDF y los datos son almacenados en un Data Warehouse(DW). Para este proyecto se ha desarrollado una ontología del dominio de ensayos clínicos sobre cáncer, llamada HDOT [4].

Las anotaciones sirven para realizar la traducción de datos, que consiste en obtener información de una base de datos heterogénea, también llamada base de datos física, y reestructurarla con al esquema conceptual, en el caso de p-medicine es HDOT.

Una de las tareas del grupo de informática biomédica (GIB), fue la implementación de una herramienta para la creación manual de anotaciones llamada *Ontology Annotator (OA)*.. Esta herramienta permite crear las anotaciones requeridas en p-medicine aunque su uso resulta complejo para usuarios con un perfil no técnico. Por ello ha sido necesario desarrollar una herramienta que complementa al OA y que tendrá como objetivo asistir a los usuarios en el proceso de la anotación, automatizando los pasos más complejos. La aplicación generará anotaciones automáticamente y mostrará estas en lenguaje natural de manera que se abstrae al usuario de la complejidad generada por el *Ontology Annotator*.

1.2 Objetivos del trabajo

El objetivo del trabajo es la realización de una herramienta que asista al usuario en la tarea de definir anotaciones basadas en vistas y genere dichas vistas de forma semiautomática.

Resulta evidente al plantear el problema, que conseguir un sistema de alineamiento automático de vistas es por ahora imposible. Este trabajo se limita a asistir a un usuario no experto en la tarea de conseguir alinear anotaciones de bases de datos.

1.2.1 Solución propuesta

La solución propuesta es una herramienta con interfáz gráfica que automatizará las tareas de *Ontology Annotator*. Sus principales funciones son:

- Descripción en lenguaje natural de vistas de modelos RDF
- Generación automática de todas las vistas del modelo físico
- Generación de la vista conceptual equivalente basándose en técnicas de similitud léxica y semántica.

La herramienta además irá mostrando una barra de progreso que haga saber al usuario la cantidad de trabajo que ha realizado hasta el momento. Adicionalmente podrá salvar el progreso de su trabajo y retomarlo en otro momento así como guardar el *Mapping* sin necesidad de llegar hasta el final del proceso.

CAPÍTULO 2
ESTADO DE LA CUESTIÓN

En este capítulo se revisarán diferentes algoritmos de similitud entre términos que serán usados en el desarrollo de este trabajo, ya que el alineamiento automático de vistas exige de la utilización de varios métodos de similitud entre términos y frases para atajar el problema. A. Escrich, 2013[5] reúne en su trabajo una serie de algoritmos que son divididos en tres grupos: Léxicos, Estructurales y Semánticos.

2.1 Algoritmos léxicos

Se centran en encontrar la similitud entre dos términos evaluando estrictamente ambas cadenas de caracteres. Si bien es cierto que no todos estos algoritmos son aplicables al problema de alineamiento de vistas, algunos de los que se describen a continuación pueden dar buenos resultados:

Distancia Hamming:

Se evalúan dos cadenas de caracteres del mismo tamaño. La distancia será el número de cambios que hay que realizar para convertir una palabra en otra posición a posición.

```
Michael
Mitchel
```
} Distancia Hamming → 3

Distancia Damerau-Levenshtein:

Mide la distancia entre dos términos en base a la cantidad de operaciones que hay

que hacer para transformar una cadena en otra. Las operaciones de inserción, eliminación y sustitución son cuantificadas como una sola operación mientras la permutación penaliza 2. Una distancia equivalente a 0 significa que ambas cadenas son la misma. La implementación de este algoritmo por A. Escrich, 2013 trata mayúsculas y minúsculas como igual, de manera que "X" equivale a "x" y por lo tanto no penalizaría como operación de sustitución. Se puede establecer un umbral en el que se entiende que una distancia mayor a esta significaría que ambas cadenas de caracteres son muy diferentes. A continuación se muestra un paso a paso para calcular la distancia entre Michael y Mitchel:

Paso 1	**Michael**	**Mitchael**	**Añadir carácter**
Paso 2	**Mitchael**	**Mitchael**	**Eliminar carácter**
	Mitchel		**Distancia → 2**

Distancia Jaro-Winkler:

Este algoritmo devuelve una distancia comprendida entre 0 y 1 entre dos términos, donde 1 es un *match* entre las dos palabras y 0 implica la mayor diferencia entre ambas. La formula para calcular esta distancia es la siguiente:

$$\frac{1}{3}\left(\frac{m}{|s_1|}+\frac{m}{|s_2|}+\frac{m-t}{m}\right)$$

Figura 1: Fórmula Jaro-Winkler

donde:

- ***m*** es el número de caracteres comunes
- ***t*** es el números de transposiciones
- ***s1*** y ***s2*** son la cantidad de caracteres de cada palabra

2.2 Algoritmos semánticos

Los algoritmos semánticos trabajan con el conocimiento y la interpretación de la cadena de caracteres. En este caso los resultados no son cuantificables como en los algoritmos léxicos. El proyecto WordNet de la Universidad de Princeton ofrece un diccionario con las diferentes relaciones entre términos:

- **Sinónimo**

Vocablo o expresión que tiene el mismo, o muy parecida significación a otro.

- **Hipónimo**

Palabra que esta englobada dentro del significado de otra. Por ejemplo quimioterapia respecto a tratamiento.

- **Hiperónimo**

Palabra que engloba el significado de otras. Tratamiento respecto a radioterapia o quimioterapia.

- **Homónimo**

Palabra que aun que escrita de igual forma que otra, tiene distinta significación.

La alineación entre sinónimos puede ser muy útil ya que aunque la distancia léxica puede ser muy alta, el significado es el mismo y por lo tanto podría interpretarse como un acierto a la hora de alinear. Sin embargo, la existencia de las homonimias (palabras que se escriben de la misma forma aunque cuando el significado es diferente) puede llevar a error. Como por ejemplo la palabra “paciente” que tiene 5 acepciones diferentes:

Paciente
1. adj. Que tiene paciencia.
2. adj. Fil. Se dice del sujeto que recibe o padece la acción del agente.
...
5. com. Persona que es o va a ser reconocida médicamente.

2.4 Alineamiento de Ontologías

El alineamiento entre dos ontologías es parte fundamental de un proceso de migración o integración de bases de datos. Existen enfoques diferentes para el alineamiento de ontologías: *elemento a elemento* y *vista a vista*.

Los basados elemento a elemento se centran en alinear términos por su similitud léxica o semántica, en cambio, los vista a vista buscan encontrar la similitud de todos los términos y relaciones implicados en la vista. Para ello es necesaria la asistencia de un experto que determine que términos han de ser alineados entre si. Existen trabajos para la detección automática de términos sueltos, pero sería más interesante dar un paso más allá y conseguir realizar el mismo trabajo dirigido a vistas.

2.5 Alineamiento de Vistas

Para entender el alineamiento de vistas se necesita estar familiarizado con los conceptos que se describen a continuación:

- **Triples**

  Es la forma de representar información de la forma sujeto-predicado-objeto. En otra palabra, la representación de dos términos unidos por una relación. Tal como se puede observar en la Figura 2.

Figura 2: Triple

- **Path**

  Grafo dirigido compuesto por una o más triples. En la Figura 3 se muestra un *Path* generado por dos triples.

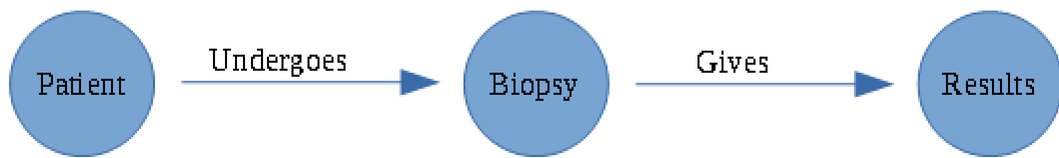


Figura 3: Path

- **Internal bounds**

Puntos de anclaje de dos *Paths* diferentes que comparten un clase en común.

- **Vista**

Conjunto de uno o más *Paths* unidos entre si por *Internal Bounds* como se puede observar en la Figura 4.

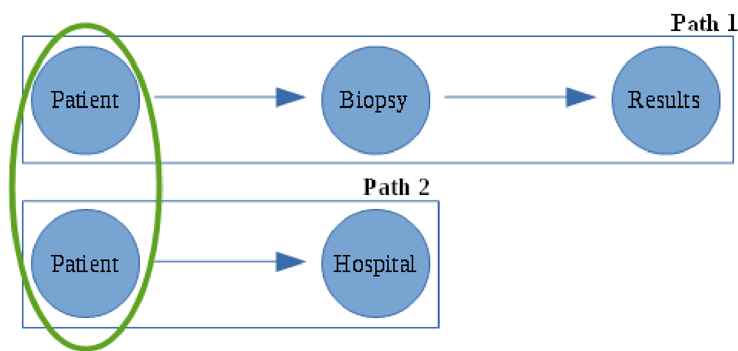


Figura 4: Vista e Internal Bound

Existen trabajos de alineamiento de vistas como la planteada en A. Anguita [6] en el que se presenta una herramienta para la generación de *Mappings*. También se menciona una herramienta llamada KARMA para la generación automática de esquemas RDF desde una estructura tabular de datos. Sin embargo, no existe ningún trabajo previo en cuanto a alineamiento automático de vistas. La herramienta que aquí se presenta pretende sentar un precedente.

CAPÍTULO 3
TECNOLOGÍAS EMPLEADAS

En este capítulo se van a exponer las tecnologías y lenguajes de programación que se han empleado para el desarrollo de la herramienta.

3.1 Lenguajes de programación

3.1.1 Java

Java es un lenguaje de programación orientado a objetos que guarda cierta similitud con otros lenguajes como C++ o C#, también orientados a objeto. Diseñado en 1991 y publicado en 1995 por Sun Microsystems (compañía que fue absorbida en 2009 por Oracle). Parte de las tecnologías de Java tienen licencia GNU[7] desde mayo del 2007 siguiendo las especificaciones de *Java Community Process.* Existen alternativa desarrolladas por terceros que son completamente abiertas tales como GCJ (GNU Compiler for Java) o GNU Classpath.

La mayor de las ventajas que Java ofrece es su independencia sobre la arquitectura o el sistema operativo de la máquina. Esto permite que una vez el código ha sido compilado, el resultado podrá ejecutarse en cualquier máquina que disponga de JVM (Java Virtual Machine), aplicación capaz de interpretar y ejecutar el código compilado de Java.

Java se ha impuesto sobre otros lenguajes orientados a objetos gracias a su versatilidad para desarrollar diferentes tipos de aplicaciones: Web, Movil, Escritorio y Servidores. Tiene ademas un gran respaldo en la comunidad de desarrolladores lo que

hace muy sencillo encontrar implementaciones de terceros que pueden ser importadas a un proyecto.

Otra de las ventajas que ofrece es la facilidad para generar documentación gracias a Javadocs mediante comentarios específicos dentro del código. Esto permite crear un manual para desarrolladores con una interfaz web.

Existen diferentes entornos de desarrollo integrado para Java entre los que se encuentra Eclipse[8] o IntelliJ[9] (una versión mas trabajada de Eclipse). Ambos son gratuitos aunque este último ofrece una versión *Ultimate* con mas funcionalidades. Es por ello que para el desarrollo de esta herramienta se ha decidido hacer uso de IntelliJ.

El entorno de ejecución se puede descargar gratuitamente desde la web[10] de Oracle. Alternativamente, si se es desarrollador, existe la posibilidad de descargar el JDK (Java Development Kit) que incluye todos los programas necesarios para desarrollar en este lenguaje.

3.1.2 JavaScript

Es un lenguaje de programación interpretado empleado generalmente en la parte del cliente, aunque existe la posibilidad de ejecutarlo en la parte servidora. La totalidad de los navegadores web son capaces de interpretar JavaScript y es por ello que es un lenguaje muy popular para generar dinamismo en las páginas web.

Al ser interpretado (el código no se compila, se ejecuta línea a línea) resulta mas complicado depurar errores en el código, generalmente porque los navegadores guardan en cache el código de manera que no tenga que ser descargado cada vez que se solicita una página. Esto hace que aunque el código haya sido modificado, el navegador siga haciendo uso del antiguo. Firefox ofrece un *kit* para desarrolladores que permite depurar errores y usar puntos de parada en el código.

Jquery[11] es una librería que ofrece una versión más simple de JavaScript.

Esencialmente se hace uso de anotaciones más básicas e implementa funciones que JavaScript no contiene. En esta herramienta Jquery se emplea para hacer llamadas asíncronas al servidor web así como modificar estilos dinámicamente.

3.2 Otros lenguajes

3.1.1 HTML

HTML (HyperText Markup Language) es un estándar para el modelado de páginas web basado en etiquetas . Es una especificación que pertenece a W3C al igual que otros lenguajes vistos en este trabajo. Cada vez más popular debido a la proliferación de aplicaciones en la nube. La última versión (Junio de 2014) es HTML5 aunque en esta herramienta no se hace uso de las nuevas funcionalidades de esta versión. Desarrollar la interfaz en este lenguaje permite que el usuario final pueda acceder desde cualquier dispositivo con un navegador web y que disponga de acceso al recurso donde esté almacenada la herramienta.

3.2.2 RDF

RDF es el lenguaje para el modelado de datos que se ha impuesto en el campo de la Biomedicina. Es una estructura jerárquica de clases y relaciones que unen estas clases. RDF permite además acoplar diferentes esquemas mejorando así la escalabilidad. La representación de la información se da en forma de triples (origen – relación – destino) y se hace uso de URIs para el etiquetado de sus componentes. De esta manera es posible unir información proveniente de fuentes diferentes. Estas triples pueden estar representadas directamente en RDF o externamente en un fichero N-triples [12], un fichero de texto simple que representa una triple por línea.

3.2.3 OWL

OWL[13] (Web Ontology Language) es un lenguaje que extiende el lenguaje RDF

añadiendo nuevas propiedades. OWL tiene 3 sublenguajes: OWL Lite, OWL DL y OWL Full.

- **OWL Lite**
  Esta versión es la que mejor se adapta a soluciones que necesitan básicamente una jerarquía de clasificación y restricciones simples.

- **OWL DL**
  Extiende la versión Lite y acepta todas las construcciones del lenguaje OWL posibles. Sin embargo tiene ciertas restricciones (Una instancia de una clase no puede ser instancia de ninguna otra).

- **OWL Full**
  Es el sublenguaje OWL que ofrece mayor libertad para modelar datos. Por ejemploOWL Full una clase puede ser tratada a la vez como una colección de individuos y como persona por derecho propio.

A continuación se enumeran las propiedad mas importantes que ofrece OWL sobre RDF:

Características de Properties (Relaciones entre clases e instancias):

- ObjectProperty
- DatatypeProperty
- inverseOf
- TransitiveProperty
- SymmetricProperty
- FunctionalProperty
- InverseFunctionalProperty

Restricciones sobre relaciones:

- Restriction

- onProperty
- allValuesFrom
- someValuesFrom

Restricciones de cardinalidad:

- minCardinality (only 0 or 1)
- maxCardinality (only 0 or 1)
- cardinality (only 0 or 1)

Existe una versión OWL2[14] publicada el en Diciembre de 2012 que añade las siguiente características a su predecesor:

- Claves
- Cadenas de propiedades
- Mas *Datatypes* y Rangos
- Mejoras en las restricciones de cardinalidad
- Propiedades asimétricas, reflexivas y dijuntas
- Mejoras en las anotaciones

3.3 APIs externas

3.3.1 MappingAPI2

API desarrollada por el GIB escrita en Java que ofrece métodos para crear Mappings basados en vistas entre fuentes RDF. Los objetos MappingAPI2 son una serie de alineaciones vista a vista cuyo resultado es un fichero XML[14] que contiene la información sobre triples, paths, enlaces internos, enlaces externos, vistas y metadatos (Descripciones, origen de los datos, fecha, autor...). Permite además generar

restricciones o hacer que ciertas clases sean identificadoras.

```
<entry>
 - <physical_view>
   - <paths>
     - <path>
         <class externalBound="ExternalBound3" internalBound="InternalBound1">http://www.sample01.owl#Biopsy</class>
         <property>http://www.sample01.owl#happensAfter</property>
         <class externalBound="ExternalBound1">http://www.sample01.owl#Chemotherapy</class>
       </path>
     - <path>
         <class internalBound="InternalBound1">http://www.sample01.owl#Biopsy</class>
         <property>http://www.sample01.owl#reveals</property>
         <class externalBound="ExternalBound2">http://www.sample01.owl#TumorSample</class>
       </path>
     </paths>
   </physical_view>
 - <conceptual_view>
   - <paths>
     - <path>
         <class externalBound="ExternalBound1" internalBound="InternalBound1">http://purl.bioontology.org/ontology/RID/RID39252</class>
         <property>http://edamontology.org/has_output</property>
         <class externalBound="ExternalBound2">http://edamontology.org/format_2001</class>
       </path>
     - <path>
         <class internalBound="InternalBound1">http://purl.bioontology.org/ontology/RID/RID39252</class>
         <property>http://edamontology.org/has_identifier</property>
         <class externalBound="ExternalBound3">http://edamontology.org/format_2352</class>
       </path>
     </paths>
   </conceptual_view>
</entry>
```

Figura 5: XML generado por el MappingAPI2

3.3.2 OWLBasicModel2

API desarrollada por el GIB y escrita en Java que ofrece métodos para la gestión de modelos RDF. Los objetos OWLBasicModel2 nos ofrecen métodos para recuperar información referente a nombres de clases y relaciones, padres de clases y relaciones o las distintas relaciones existentes entre dos clases.

3.3.3 SemanticEquivalence

API desarrollada por el GIB y escrita en Java que tiene implementados diferentes algoritmos de comparación de términos para medir la similitud entre ellos. Los algoritmos implementados se listan a continuación:

- Distancia Hamming
- Distancia Levenshtein
- Distancia Damerau-Levenshtein
- Distancia Jaro-Winkler
- Distancia Needleman-Wunsch
- Distancia N-Gram

3.3.4 JAWS (Java API for WordNet Searching)

API desarrollada por la Universidad de Lyle y escrita en Java que ofrece métodos que explotan el diccionario WordNet para obtener sinónimos, hipónimos, hiperónimos, homonimos y otro tipo de relaciones semánticas entre términos. En principio la API es independiente del proyecto Wordnet lo que permite actualizar el diccionario sin necesidad de cambiar la API. Cabe destacar que solo está disponible en inglés.

3.3.5 JDOM

JDOM[16] es una biblioteca de código abierto escrita en Java que permite la manipulación y generación de documentos en XML. Aunque tiene muchas similitudes con DOM[17] no se adaptada del todo a la especificación. El principal motivo es que DOM fue diseñado para páginas web escritas en HTML y Javascript mientras JDOM es una adaptación a las necesidades especificas de Java tales como la sobrecarga de métodos. En esta API se emplea para salvar o cargar desde un fichero XML el progreso del trabajo realizado.

```
<saved_progress>
 - <view>
     <action name="ACCEPT_VIEW"/>
   </view>
 - <view>
     <action name="SPECIFY_VIEW_INT_STRING" parameters="0##http://www.sample01.owl#happensAfter"/>
     <action name="ACCEPT_VIEW"/>
   </view>
 - <view>
     <action name="SET_IDENTIFIER_INT" parameters="1"/>
     <action name="SET_RESTRICTION_INT_INT_RESTRICTION" parameters="0##1##LESS##ssds##STRING"/>
     <action name="ACCEPT_VIEW"/>
   </view>
   <view current="true"/>
</saved_progress>
```


Figura 6: XML del progreso almacenado

3.4 Software empleado

- **Apache Tomcat**

  Apache Tomcat[18] es un servidor web de código abierto así como un contenedor de servlets. Está desarrollado en Java, lo que permite instalarlo en cualquier máquina independientemente de su arquitectura o sistema operativo. Esencialmente es un proyecto que adapta Apache (El servidor web más popular) para poder interaccionar con APIs Java corriendo en el servidor. La versión que se debe instalar en la máquina servidora para poder desplegar el proyecto será la 7 ya que es la que es 100% compatible con la versión usada para el desarrollo del *Ontology Annotator Assistant.*

- **Prótegé**

  Prótegé[19] es la herramienta por excelencia para trabajar con ontologías desarrollada por la universidad de Stanford en colaboración con la universidad de Manchester. Está desarrollada en Java lo que permite ejecutarlo en cualquier

plataforma. Actualmente (Junio de 2014) tiene mas de 244817 usuarios, lo que lo ha convertido en el software referente para el manejo de estas estructuras de datos.

En ocasiones trabajar con ficheros RDF y OWL en texto bruto puede resultar complicado y caótico. Esta herramienta permite que terceros desarrollen *plugins* análogamente a otros productos tales como Eclipse. La versión descargable desde la web de Stanford permite visualizar las fuentes RDF como grafos dirigidos.

La web del proyecto ofrece una aplicación web llamada WebPrótegé más elemental que la versión de escritorio y compleja de usar. En el mismo proyecto pueden encontrar ontologías desarrolladas por terceros.

CAPÍTULO 4
MÉTODOS

La aplicación hace un doble trabajo a la hora de mostrar en pantalla los resultados. Por una parte genera de forma totalmente automática vistas de la base de datos física, y permite al usuario interaccionar con las clases y términos pertenecientes a estas vistas. Por otra parte, la aplicación trabajará para dar como resultado la vista conceptual más semejante automáticamente. En ambos casos, las vistas RDF mostradas al usuario se presentan en forma de lenguaje natural, para facilitar la comprensión de las mismas y posibilitar que la herramienta sea empleada por usuarios con un perfil no técnico.

4.1 Algoritmo para creación de vistas de la base de datos física

Las estructuras de datos basadas en RDF pueden ser interpretadas como grafos dirigidos. Para la generación de paths posibles en la parte física nos valdremos de esta propiedad recorriendo todo el grafo en profundidad. De manera que el algoritmo pueda generar todos los subconjuntos posible del esquema. El recorrido del esquema físico puede verse alterado cuando se toman ciertas decisiones tales como generalizar o especificar. A continuación se describe el algoritmo detalladamente:

1. **Preparación previa al recorrido**

El algoritmo busca en el modelo físico todos los nodos raíz del esquema. De esta forma se asegura que se generarán todos los subconjuntos posibles. Se seleccionará uno de estos nodos y el algoritmo estará preparado para recorrer el grafo en profundidad.

Cuando haya recorrido todo el árbol se procederá de la misma forma con el siguiente nodo raíz, si este existiera.

2. **Generando subconjuntos**

Partiendo de la raíz, el algoritmo generara subconjuntos de longitud 1 (2 clases unidas por una relación), el menor tamaño posible de una vista en RDF. El algoritmo se sitúa en un nodo y recorre todas las relaciones que salen de él hacia otros nodos. Al terminar de recorrer todas properties avanzará en una de estas si fuera posible, de lo contrario se sube un nivel y se comienza a iterar de nuevo sobre el siguiente nodo.

En el caso en que el modelo lo permita, el usuario podrá modificar la longitud de estos subconjuntos y para ello puede tomar dos decisiones:

- **Especificar**

Se concreta más la vista y se incrementa la longitud del subconjunto que se ha generado. En este caso se alarga la vista recorriendo las relaciones de la última clase del subconjunto o de la clase seleccionada por el usuario (lo que generaría una nueva rama). Adicionalmente si no le interesara recorrer todas las relaciones que parten de este nodo, podrá concretar cuál de las relaciones quiere recorrer.

- **Generalizar**

El usuario necesita una vista más general, con menos información, por lo que la vista se acorta. Generalizando, la primera clase desaparece para que el subconjunto parta desde la que estaba en segunda posición. El usuario podrá también concretar sobre que clase quiere generalizar la vista.

4.1.1 Caso práctico de recorrido del esquema físico

Para mostrar el funcionamiento de este algoritmo haremos uso de un esquema que

contempla todas los casos que pueden darse en el recorrido.

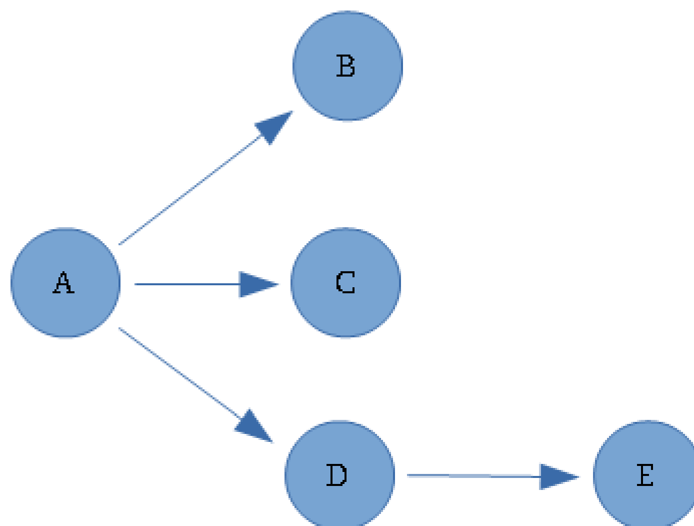


Figura 7: esquema RDF representado mediante un grafo dirigido

La clase A es el único nodo raíz que tiene el esquema. Será esta clase por tanto el punto de partida para recorrer el árbol. Partiendo de este nodo podrán generarse subconjuntos de longitud 1 con las clases B,C y D.

El subconjunto de longitud 1 que tiene como origen A y como destino D es la única vista de este esquema que es especificable, ya que desde D sale una propiedad que lleva a E. Si se especificara esta vista, se alargaría partiendo de D.

Si el usuario quisiera especificar la vista en base a una relación concreta este solo podría hacerlo sobre las clases A y D ya que son las única que tiene relaciones que llevan a otros nodos. En este caso el algoritmo genera una nueva rama (no alarga la vista).

En este esquema solo sería generalizable la vista A → D → E, que es la vista con mayor longitud que se puede crear con este esquema RDF. Como resultado de generalizar esta vista, el resultado quedaría de la forma D → E.

4.2 Algoritmo de alineamiento automático

Cogiendo como entrada las vistas generadas en la parte física, el algoritmo de alineamiento automático construye la vista más verosímil en la parte conceptual. La solución propuesta e implementada en el *Ontology Annotator Assistant* hace uso de los métodos de alineamiento automático (léxicos y semánticos) de términos simples.

El algoritmo comenzará creando una estructura de similitudes entre ambos modelos. Esta similitud viene determinada por los resultados que nos devuelve el algoritmo para el calculo de distancia Damerau-Levensthein. Con la estructura creada, el algoritmo automático empieza a tomar decisiones en cuanto a las triples que generará en la parte conceptual en base a las distancias totales calculadas.

A continuación se describe el algoritmo el paso a paso:

1. **Generación de una estructura de similitud**

   A cada clase perteneciente al modelo físico se le asignará una lista ordenada de menor a mayor peso de clases equivalentes que se encuentran en el modelo conceptual. Este peso se genera mediante el algoritmo de Damerau-Levenshtein. Un peso equivalente a 0 significaría que tenemos un *match* en la parte conceptual.

   Adicionalmente, para cada clase de la parte física se buscan sinónimos en el diccionario de WordNet. Generalmente los términos que son sinónimos tienen una distancia Damerau-Levenshtein muy alta debida a su diferencia léxica, sin embargo, por su equivalencia semántica a estos sinónimos se les dará un peso equivalente a 0.

   En el caso de las relaciones, ya sean *object properties* o *data type properties,* se procederá de la misma forma que con las clases.

2. **Generación de todas las posible triples de la parte conceptual**

Como punto de partida se genera un listado de todos las clases que se encuentran en la parte física. A partir de aquí se construye una lista de triples en la parte conceptual con los términos más similares para cada una de las clases en la parte física. A cada una de estas triples conceptuales se le asigna una distancia. Esta distancia se utiliza para ordenar la lista de triples de menor a mayor ya que se entiende que la triple más acertada es aquella que tiene una menor distancia en sus términos y la relación.

En este paso se hace uso de la estructura generada en el paso anterior. La parte física deja de ser importante ya que similitud ya está generada. Se entiende que para cada término en la parte física, tenemos un equivalente en la parte conceptual. Con estos equivalentes se generan todas las triples posibles "todos a todos".A cada una de estas triples se le asignara una distancia que será la suma de las distancias Damerau-Levenshtein de ambos términos más la distancia Damerau-Levenshtein de la relación que las une.

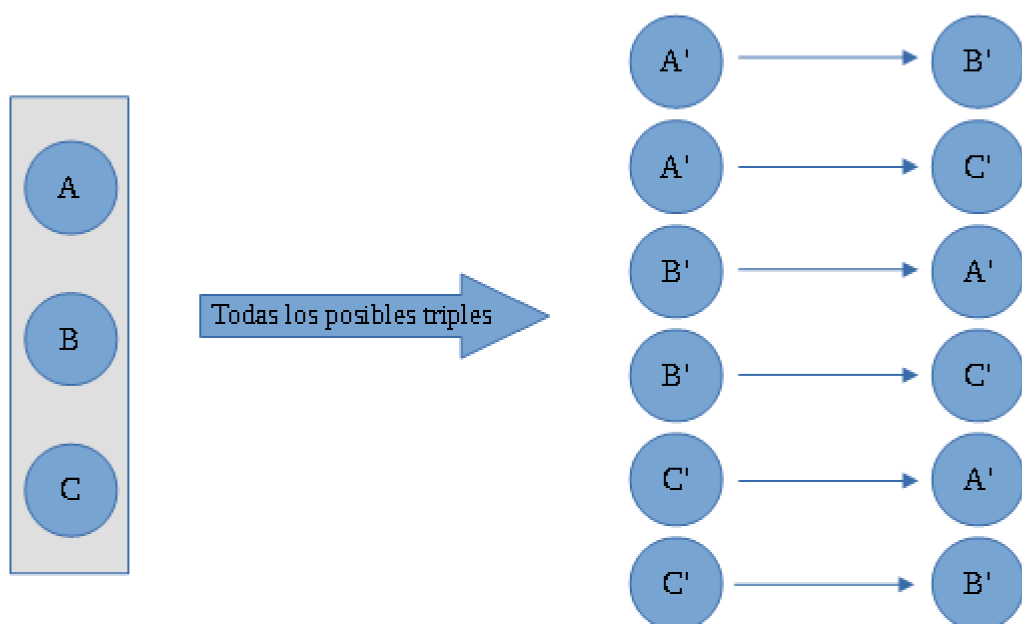


Figura 8: De 3 clases físicas salen 6 triples en la parte conceptual

A medida que el modelo físico tenga mas clases, la cantidad de triples que se

pueden generar en la parte conceptual crecen de la forma (n-1)*n como se muestra en la Figura 8.

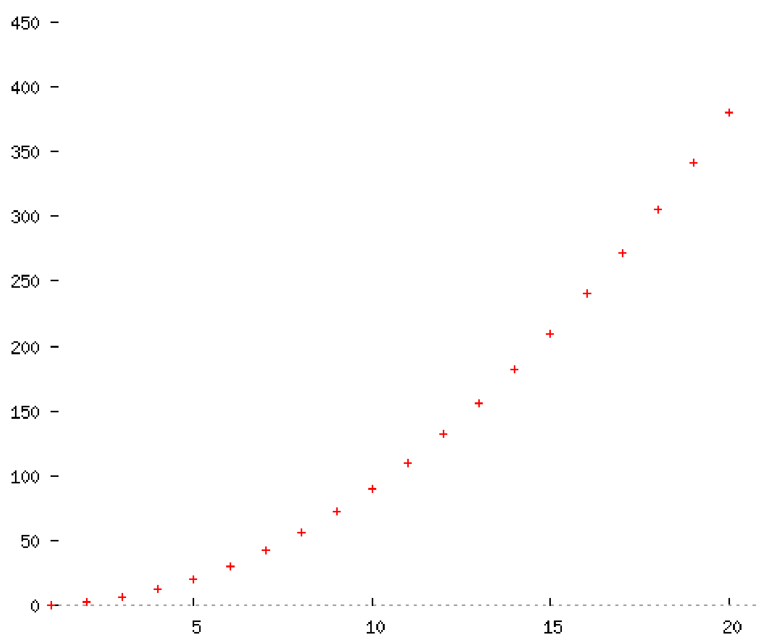


Figura 9: Crecimiento (n-1)*n de las triples que se pueden generar.

3. Generación de una vista conceptual con las triples de menor peso

Con la pila de triples generada en el paso anterior se procede a construir la vista en la parte conceptual. Cada vez que se desapila una triple pueden darse 4 condiciones diferentes:

- **Ambos términos de la triple ya han sido añadidos a la vista conceptual**

  En este caso la triple se desecha ya que no tiene sentido volver incluir ambos términos de nuevo. En alguno casos esto llevaría a crear ciclos dentro del grafo dirigido. En los casos en la que ya se han añadido todos los términos a la vista, pero esta no es conexa, entonces se usará esta triple para unir las ramas.

- **El origen de la triple ya ha sido incluido en la vista conceptual anteriormente**

  Si el origen de la triple ya había sido incluido con anterioridad solo haría falta añadir la triple, o bien al final de una rama o bien como una nueva rama.

- **El destino de la triple ya ha sido incluido en la vista conceptual anteriormente**

  Si el destino de la triple ya había sido incluido con anterioridad solo haría falta añadirla, o bien al principio de una rama o bien como una nueva rama.

- **Ni el origen ni el destino se han añadido anteriormente a la vista**

  El caso más sencillo ya que esto solo implica crear una nueva rama. El siguiente diagrama de flujo detalla el funcionamiento de este tercer paso.

4.2.1. Diagramas de flujo del algoritmo automático

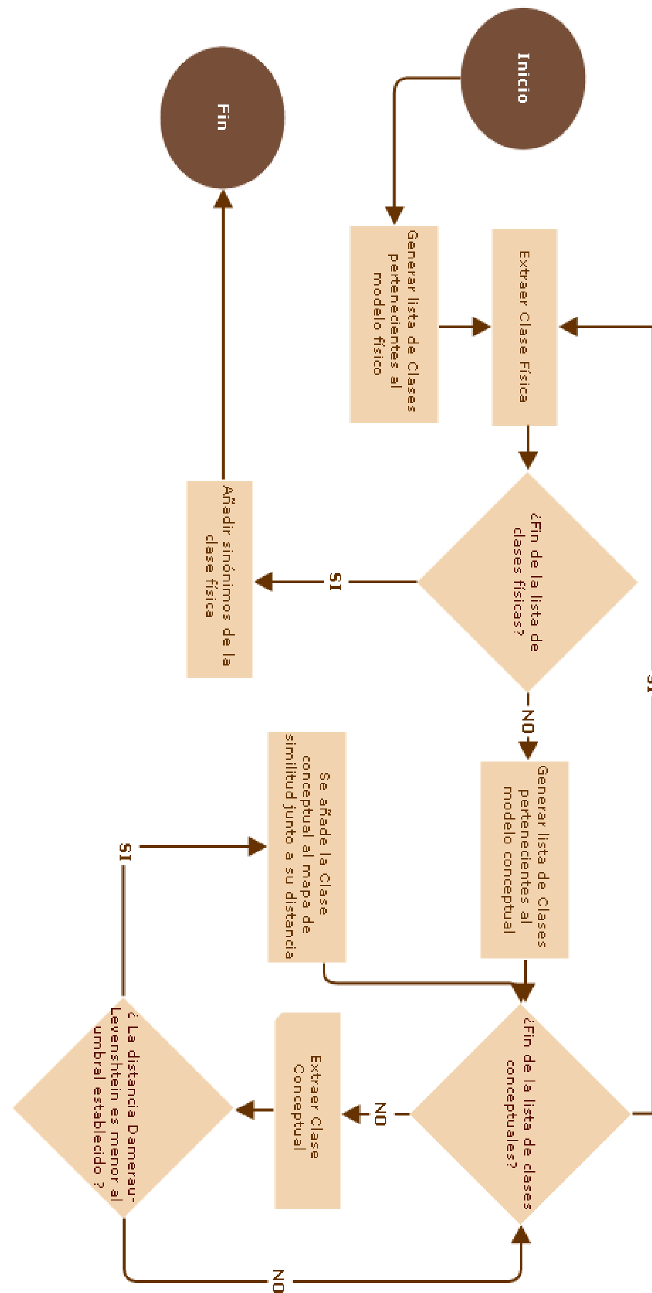


Figura 10: Generación de una estructura de similitud

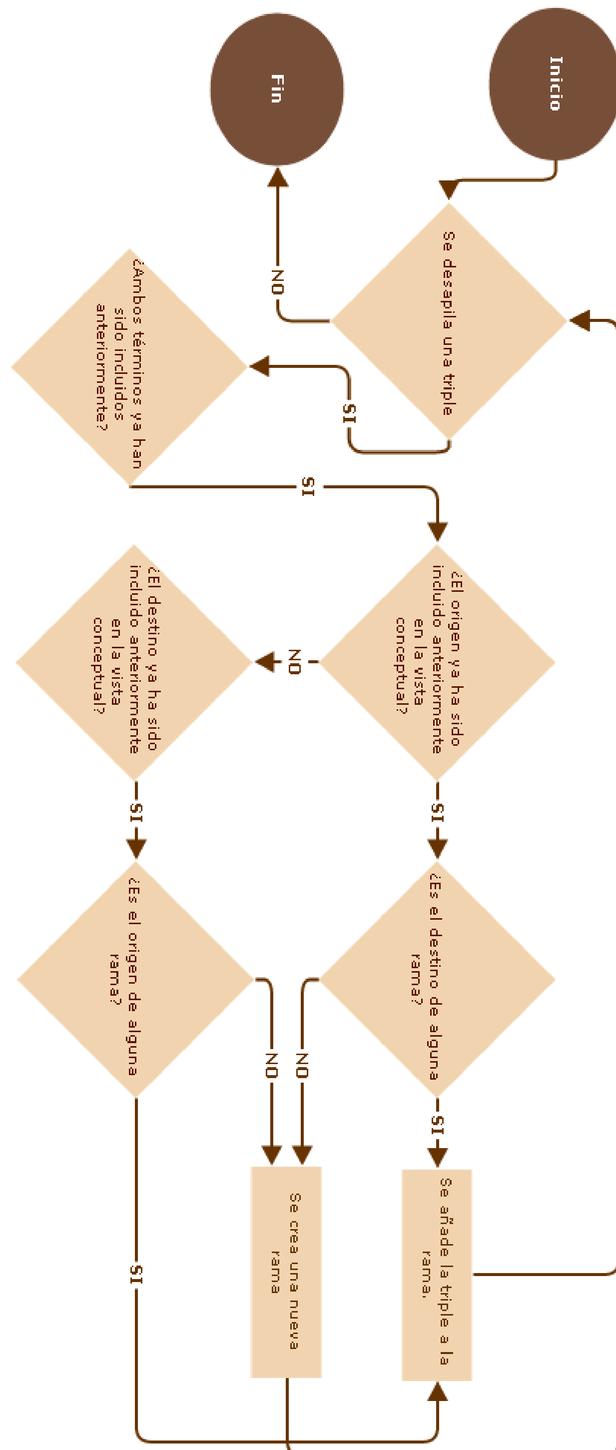


Figura 11: Generación de una vista conceptual con las triples de menor peso

4.3 Construcción de frases en lenguaje natural a partir de vistas RDF

RDF permite que las clases y relaciones puedan ser etiquetadas de manera que no haya que hacer referencia a la URI entera. Esto permite que las triples, y por extensión las vistas, puedan interpretarse como frases en lenguaje natural. No obstante es necesario añadir ciertos conectores para que las frases construidas tengan una estructura gramatical correcta. Debido a que las clases y propiedades proporcionadas en bases de datos biomédicas están en ingles se ha decidido que lenguaje natural mostrado en la aplicación sea también en este idioma.

En la primera de las triples de una vista, entre el sujeto y el predicado se añade la palabra "that". Las triples siguientes contendrán en la misma posición la palabra "which". En el caso en el que alguna de las clases fuera identificadora, al final de la frase se añade el texto "The **nombre de la clase** act as identifier.".

La herramienta permite añadir restricciones sobre las clases y estas también se muestran en lenguaje natural. Para ello la se genera una frase que sigue el patrón "The **nombre de la clase** must be **operador restricción".** Donde el operador es un enumerador que puede tener los siguientes:

- equal to
- different from
- less than
- less or equal than
- greater than
- greater or equal than

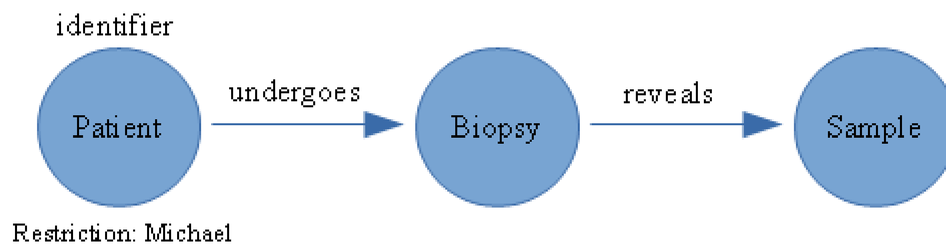


Figura 12: Vista RDF de un solo path con dos triples

Para la vista mostrada en la Figura 12 se generaría la siguiente frase en lenguaje natural:

Patient(s) that undergoes Biopsy, which reveals Sample. The Patient must be equal to "Michael". The Patient(s) act as identifier

CAPÍTULO 5
ANÁLISIS DEL SISTEMA

5.1 Características de la herramienta

A continuación se procede a describir las diferentes funcionalidades que la herramienta ofrece. La interfaz de la aplicación consta de una sola pantalla con la que interaccionar como se muestra en la figura Figura 13.

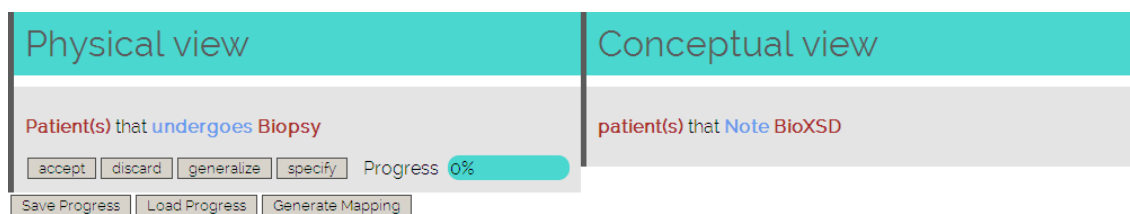


Figura 13: Pantalla principal de la herramienta

La pantalla se divide en dos, la mitad izquierda representa la vista física y la mitad derecha la vista conceptual. Toda interacción con la aplicación se realiza siempre sobre la parte física. La herramienta irá recorriendo la base de datos y mostrando en pantalla mediante lenguaje natural las vistas generadas. En la parte derecha a la misma altura se mostrará el resultado más verosímil para la parte conceptual.

Las acciones que puede realizar el usuario se pueden dividir en dos grupos: Acciones realizadas sobre la vista y acciones realizadas sobre los términos y relaciones.

Acciones sobre la vista

Bajo las frases generadas por la herramienta se muestran cuatro botones (Accept,

Discard, Generalize, Specify). A continuación se detalla el funcionamiento de cada una:

- **Accept**

  Las vistas mostradas en pantalla serán incluidas en el resultado final.

- **Discard**

  Las vista mostradas en pantalla serán obviadas en el resultado final

- **Generalize**

  Simplifica o acorta la vista que se muestra en pantalla.

- **Specify**

  Alarga la vista desde el último término mostrado en la frase.

Acciones sobre términos y relaciones

Para realizar una acción específica sobre alguno de los términos (en color rojo) o relaciones (en color azul) es necesario hacer click sobre el. Un nuevo recuadro se mostrará con las acciones posibles tal como se muestra en las figuras 5.2 y 5.3.

Figura 14: Detalle del recuadro de acciones para un término.

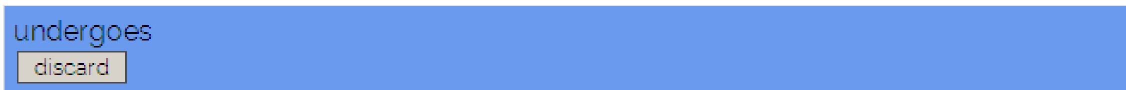


Figura 15: Detalle del recuadro de acciones para una relación.

En la parte superior derecha se muestra el nombre del término o relación sobre la que se va a actuar. En el caso de los términos, bajo el nombre se muestra una lista de sinónimos sacados de la API JAWS Wordnet [20]**.** El campo de texto bajo esta lista permite añadir nuevos sinónimos manualmente en el caso de que fuera necesario. La **x** al principio de cada sinónimo permite eliminarla de la lista.

Generalize

Simplifica o acorta la rama donde se encuentra el término.

Specify

Genera una nueva rama partiendo del término sobre el que se realiza la acción o alarga la rama a partir de este término.

Discard

Se descarta el término o la relación de y no será incluida en futuras vistas.

Identifier

El término se convierte en identificador.

Set Restriction

Un desplegable muestra los diferentes operadores que se pueden aplicar a la

restricción ($<$, $\leq$, $=$, $\neq$, $\geq$, $>$). En el campo de texto adyacente se escribirá el valor de la restricción.

A medida que se va avanzando en el proceso la barra de progreso incrementa, de manera que el usuario tiene una referencia de cuanto trabajo ha realizado hasta el momento. Cuando el usuario lo estime podrá guardar o retomar dicho progreso mediante los botones “Save Progress” y “Load Progress”. Además podrá generar el fichero de Mapping en todo momento pulsando el boton “Generate Mapping”.

5.2 Especificación de requisitos del software

5.2.1 Introducción

En este capítulo se recoge la Especificación de Requisitos software (ERS) así como la documentación correspondiente al análisis de la herramienta. La organizaciónn y formato de esta sección son los recomendados en el estándar “IEEE Recommended Practice For Software Requirements Specications" (IEEE Std. 830-1998).

Para identificar cada requisito de forma única se han numerado con el siguiente formato:

[REQ.#XX]: Descripción del requisito en lenguaje natural.

Donde el campo marcado por **X** representa la numeración, correlativa y creciente de cada requisito.

Este documento se dirige en primer término a los desarrolladores que puedan necesitar conocer los requisitos software del sistema para su ampliación u optimización y por otro lado al usuario final para que conozca los objetivos del sistema.

5.2.1.1 Propósito

El objetivo de esta sección es listar los requisitos de software de la herramienta que se ha desarrollado. Las siguientes líneas serán una guía para que el desarrollador pueda conocer las características y restricciones del sistema.

5.2.1.2 Ámbito del sistema

La herramienta (Ontology Annotator Assistant) tendrá como objetivo la generación automática de anotaciones de fuentes RDF. La herramienta tendrá una interfaz web que recorrerá todas las vistas posibles del esquema de la base de datos física a la vez que mostrará su anotación más verosímil en la base de datos conceptual. Para abstraer al usuario de la complejidad del paradigma RDF, las vistas generadas automáticamente serán mostradas en lenguaje natural. De esta manera el usuario no tiene por qué estar familiarizado con los conceptos RDF y anotaciones.

Con este trabajo se pretende que usuario sin experiencia en el modelado de datos pero con conocimientos en el campo de biomedicina puedan llevar a cabo la tarea de alineamiento de ontologías basado en vistas, un concepto que no tiene precedente.

5.2.1.3 Acrónimos y Abreviaturas

ACRÓNIMO	DESCRIPCIÓN
IEEE	Institute of Electrical and Electronics Engineers
ERS	Especificación de Requisitos Software
RDF	Resource Description Framework
HDOT	The Health Data Ontology Trunk
API	Application Programming Interface
GIB	Grupo de Informatica Biomedia (Univesidad Politécnica de Madrid)
OAA	Ontology Annotator Assistant
JAWS	Java API for WordNet Searching

Tabla 1: Acrónimos propios a la especificación de requisitos

TÉRMINO	DEFINICIÓN
Ontology Annotator	Aplicación desarrollada por el GIB que permite realizar anotaciones de fuentes RDF manualmente.
Vista	Conjunto de Paths unidos por enlaces internos
Relación	Unión entre dos clases
Mapping	Fichero resultado del alineamiento de vistas
P-medicine	Proyecto Europeo donde se engloba esta la herramienta desarrollada.
Tomcat 7	Servidor Web y contenedor de servlets desarrollados en Java.
Wordnet	Diccionario en de términos en Ingles
Javadocs	Sistema de documentación para la descripción de recursos de una aplicación escrita en Java.

Tabla 2: Definiciones propias a la especificación de requisitos

5.2.1.4 Referencias

Este apartado de Especificación de Requisitos se rige estrictamente a las recomendaciones detalladas en “IEEE Recommended Practice For Software Requirements Specications" (IEEE Std. 830-1998).

5.2.1.5 Visión General

El apartado de especificación de requisitos de software se puede dividir en tres secciones:

- **Introducción**
  Este apartado proporciona una visión global de la ERS.

- **Descripción Global**
  En este apartado se describen los factores que afectan a la herramienta y a la ERS. Se detallarán las funciones que deberá llevar a cabo este software así como las restricciones, supuestos y dependencias que sean aplicables a la herramienta.

- **Requisitos Específicos**

  En este apartado se listan todos los requisitos que el sistema debe cumplir y se proporciona toda la información necesaria para su desarrollo.

5.2.2 Descripción Global

A continuación se describe detalladamente el contexto de los requisitos, las funciones que la herramienta debe realizar, las restricciones y otros factores que intervienen en el desarrollo de este software.

5.2.2.1 Perspectiva del producto

En principio es una herramienta independiente que puede ser empleada en otros contextos y campos diferentes a la biomedicina. Sin embargo se desarrolla con la idea de ser integrado en el *Ontology Anotator*, aplicación que forma parte del proyecto p-medicine.

5.2.2.2 Funciones del producto

Las funciones que debe implementar la herramienta son las siguientes:

- Guiar al usuario mientras recorre las vistas generadas en la base de datos física
- Generar la vista conceptual más verosímil
- Descartar vistas, clases y relaciones
- Generalizar vistas y clases
- Añadir o eliminar sinónimos para las clases
- Hacer que una clase sea identificadora
- Añadir restricciones sobre clases

- Aceptar las vistas generadas
- Salvar y Carga el progreso del trabajo
- Guardar el resultado final (Generar Anotaciones)
- Mostrar Vistas RDF en lenguaje natural

5.2.2.3 Características del usuario

La herramienta no necesita de mantenimiento ni configuración por parte ningún usuario y el único perfil que se presenta es el del usuario que necesita realizar un trabajo de alineamiento de ontologías basado en vista.

El perfil del usuario que hará uso de esta herramienta es el de una persona que no tiene conocimientos sobre modelado de datos o paradigma RDF. En el *Ontology Annotator* muchos usuarios son reacios a realizar el trabajo debido a que les resulta muy complicado o no comprenden su funcionamiento. La herramienta que se presenta en este trabajo tiene como fin mostrar en lenguaje natural las vistas generadas lo que permite simplificar el perfil del usuario. Sin embargo el usuario sigue necesitando conocimientos en el campo de la biomedicina para poder hacer uso de la herramienta.

El usuario no ha de tener un perfil técnico en informática aunque tiene que estar familiarizado con la interacción web. Además este podrá hacer uso de cualquier sistema operativo y navegador web ya que esta herramienta será multiplataforma.

El modelo conceptual HDOT utilizado en la herramienta tiene términos en inglés que no serán traducidos por este software. Además para la construcción de frases en lenguaje natural se ha elegido por convenio el mismo idioma, por lo que el usuario debería tener conocimiento del inglés así como un vocabulario técnico en el campo de la biomedicina.

5.2.2.4 Restricciones

Como se ha comentado anteriormente OAA es una solución con una interfaz web por lo que será necesario que el dispositivo con el que se quiera trabajar tenga instalado un navegador web. La aplicación podrá ser utilizada en diferentes navegadores (Firefox, Opera, Explorer, Safari) instalados en cualquier sistema operativo.

La herramienta estará alojada en un servidor. Al tener una interfaz web y el código escrito en Java, será necesario que este servidor tenga Tomcat 7 instalado, ya que es esta versión la que soporta aplicaciones web desarrolladas en Java 7.

El dispositivo desde el que se acceda a la aplicación debería tener conexión a Internet y acceso a los recursos de p-medicine.

5.2.2.5 Suposiciones y dependencias

El software hace uso de APIs externas, algunas de las cuales han sido desarrolladas por el GIB. Existe una dependencia sobre estas APIs aunque un cambio en alguna de ellas no debería afectar al funcionamiento de la herramienta. Un futuro cambio a una versión superior de Tomcat no sería crítico ya que se entiende que las nuevas versiones deben soportar versiones anteriores de Java.

La aplicación hace uso de sinónimos sacados de una API desarrollada por la universidad Lyle llamada JAWS explota el diccionario Wordnet (Universidad de Princeton). Mientras el cambio en el diccionario no sea crítico, la API puede seguir obteniendo datos de el. Si se cambiara la estructura del diccionario sería necesario actualizar la versión de JAWS.

La herramienta tiene que hacer una gestión de los recursos lo más eficiente posible, ya que en ocasiones las ontologías pueden ser muy extensas y entorpecer el eficacia de la aplicación. El rendimiento de la aplicación dependerá en mayor parte del servidor donde esté alojado.

5.2.3 Requisitos Específicos

En esta sección se listarán los requisitos no funcionales que la aplicación deberá satisfacer. Todos los requisitos que se muestran a continuación son de implementación obligatoria ya que la falta de alguna de ellas no cumpliría con las necesidades planteadas para el desarrollo de la herramienta. El listado de requisitos se ha confeccionado en base a las necesidades planteadas por los usuarios de la aplicación *Ontology Annotator*.

5.2.3.1 Requisitos de Interfaces Externos

Los requisitos mostrados en este apartado se limitan al interfaz de usuario, interfaz con otros sistemas (hardware y software) así como de interfaces de comunicaciones.

5.2.3.1.1 Interfaz de usuario

- **[REQ.#01]**: La aplicación tendrá una interfaz web y su interacción se realizará medio de ratón y teclado.

- **[REQ.#02]**: La interfaz será sencilla e intuitiva y contará de una sola pantalla.

- **[REQ.#03]**: El texto mostrado en lenguaje natural estará escrito en Inglés así como los botones y demás campos de interacción.

- **[REQ.#04]**: La interfaz ha de ser flexible con las diferentes resoluciones que ofrecen las estaciones de trabajo.

- **[REQ.#05]**: La interfaz debe ofrecer *feedback* cuando está trabajando mediante mensajes de “cargando” de manera que el usuario no tenga la sensación de bloqueo en la aplicación.

5.2.3.1.2 Interfaz hardware

- **[REQ.#06]**: La aplicación web podrá desplegarse en cualquier máquina independientemente de su arquitectura o plataforma hardware.

5.2.3.1.3 Interfaz software

- **[REQ.#07]**: Tanto la parte de usuario como la parte programatica (parte del servidor) deberán ser independientes del sistema operativo.

- **[REQ.#08]**: La interfaz web deberá poder visualizarse en cualquier de los navegadores (Firefox, Chrome, Opera, Explorer, Safari).

5.2.3.1.4 Interfaz de comunicación

- **[REQ.#09]**: El sistema hará uso de los protocolos de comunicación por Internet.

5.2.3.2 Requisitos Funcionales

En este apartado se listan las funciones que debe realizar el programa para su correcto funcionamiento. Estos requisitos han sido divididos en grupos diferentes de acciones que podrá realizar el usuario.

5.2.3.2.1 Acciones sobre vistas

- **[REQ.#10]**: El usuario podrá aceptar las vistas.

- **[REQ.#11]**: El usuario podrá descartar las vistas.

- **[REQ.#12]**: El usuario podrá generalizar las vistas

- **[REQ.#13]**: El usuario podrá especificar una vista

5.2.3.2.2 Acciones sobre clases y relaciones

- **[REQ.#14]**: El usuario podrá añadir sinónimos sobre las clases
- **[REQ.#15]**: El usuario podrá eliminar sinónimos sobre las clases
- **[REQ.#16]**: El usuario podrá especificar una clase
- **[REQ.#17]**: El usuario podrá hacer que una clase sea identificadora
- **[REQ.#18]**: El usuario podrá añadir una restricción sobre las clases
- **[REQ.#19]**: El usuario podrá descartar clases y restricciones
- **[REQ.#20]**: El usuario podrá generalizar una clase

5.2.3.2.3 Progreso y Resultado

- **[REQ.#21]**: El usuario podrá guardar el progreso del trabajo realizado
- **[REQ.#22]**: El usuario podrá cargar un progreso guardado con anterioridad
- **[REQ.#23]**: El usuario podrá generar el fichero de anotaciones

5.2.3.3 Requisitos de rendimiento

En este apartado se detallan los requisitos referentes a la carga del trabajo y tiempo de espera del usuario.

- **[REQ.#24]**: El sistema generará vistas conceptuales en el menor tiempo

posible.

- **[REQ.#25]**: El sistema mostrará mensajes de espera para que el usuario no tenga sensación de bloqueo, especialmente a la hora de generar modelos de ontologías muy pesadas.

5.2.3.4 Restricciones de diseño

Las restricciones que se listan a continuación son las impuestas por estandares, plataformas hardware y plataformas software.

- **[REQ.#26]**: El sistema manejará las ontologías mediante objetos OWLBasicModel2.

- **[REQ.#27]**: Los ficheros de anotaciones serán generados mediante objetos MappingAPI2.

- **[REQ.#28]**: Los sinónimos para las clases serán extraídos de Wordnet

5.2.3.5 Atributos del Sistema

Esta sección describe propiedades que miden la calidad de la aplicación:

- **Seguridad**
- **Fiabilidad**
- **Mantenibilidad**
- **Portabilidad**

5.2.3.5.1 Seguridad

La información con la que trabaja esta aplicación no es de carácter personal ni confidencial por lo que no necesario tomar medidas en cuanto al

manejo anónimo de datos.

La identificación de usuario en OAA no es necesaria ya que la herramienta estará integrada en el *Ontology Annotator*. Esta última está restringida a usuarios con credenciales y derechos sobre los recursos que ofrece el proyecto P-medicine. La seguridad referente a ataques también se hereda de este proyecto.

5.2.3.5.2 Fiabilidad

La herramienta está diseñada de manera que varios usuarios puedan trabajar simultáneamente con la aplicación. En este caso la gestión de sesiones está implementada por Apache Tomcat, uno de los servidores web más estables.

5.2.3.5.3 Mantenibilidad

El software será implementado siguiendo las convenciones de programación establecidas por la comunidad de desarrolladores de manera que terceros puedan comprender y retomar el desarrollo del mismo. La aplicación tendrá Clases y Enumeradores bien estructuradas de manera que se facilite su escalabilidad y la implementación de futuras funcionalidades.

El código estará bien comentado de manera concisa de forma que se facilite el entendimiento y familiarización a futuros desarrolladores. Adicionalmente se escribirán Javadocs de todas las Clases y Enumeradores.

5.2.3.5.4 Portabilidad

La aplicación será desarrollada de forma que sea totalmente independiente de la plataforma hardware y software tanto en la parte usuario como en la parte programática. Para ello se harán uso de tecnologías y lenguajes de programación

que sean capaces de trabajar indistintamente de su entorno.

La parte programática será desarrollada en Java gracias al respaldo de una empresa como Oracle tiene asegurado el soporte para distintas arquitecturas de hardware y sistemas operativos. Esto facilita el tarea de volver a desplegar el proyecto en una posible migración de hardware.

La interfaz de usuario será escrita en HTML y JavaScript al ser estos lenguajes que la totalidad de estaciones de trabajo pueden interpretar.

5.3 Casos de uso del Sistema

En esta sección se detallan los actores y los casos de uso que están implicados en la aplicación. Con esta técnica podremos analizar mejor los requisitos software desde el punto de vista del usuario. Un caso de uso es la descripción de una serie de pasos o acciones que deben llevarse a cabo para realizar una tarea.

5.3.1 Actores

Los participantes en la interacción con el sistema son llamados actores. En el caso de OAA existe una única figura que se describe a continuación:

- **Usuario de anotaciones**

Al usuario se le mostrarán una serie de vistas físicas en lenguaje natural y podrá tomar decisiones sobre estas así como sobre las clases y relaciones. Podrá también guardar, cargar y finalizar el progreso de su trabajo cuando proceda.

5.3.2 Diagrama de Casos de Uso

Los diagramas de casos de uso nos permiten visualizar de forma gráfica y sencilla el comportamiento y la interacción entre actor y sistema. El diagrama que se muestra a

continuación detalla la acciones que el actor puede realizar sobre el sistema.

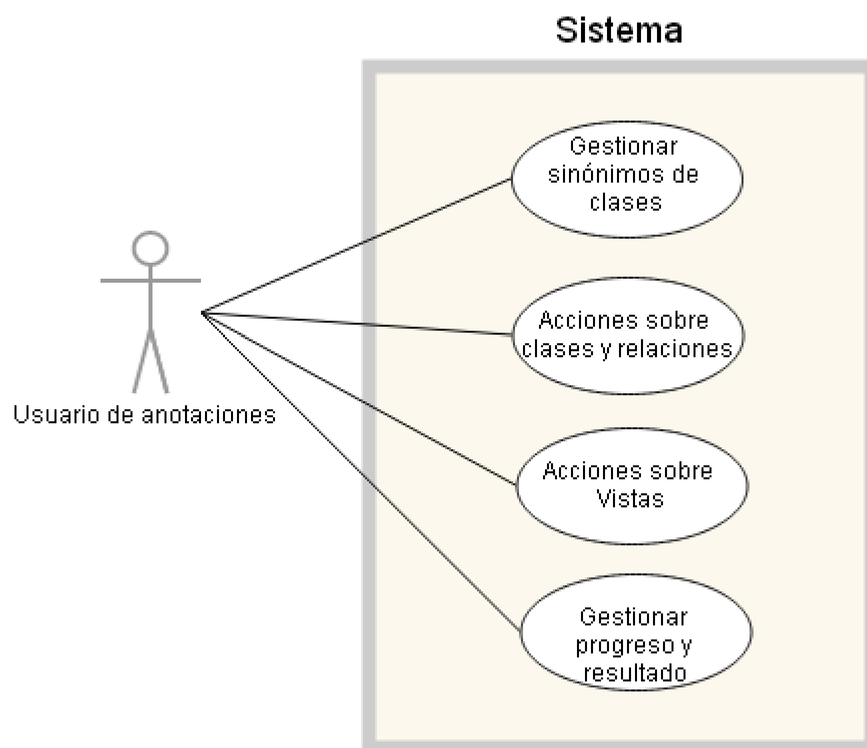


Figura 16: Diagrama de Casos de Uso del Sistema

Los casos de uso identificados en el sistema corresponden con los requisitos establecidos anteriormente.

5.3.3 Casos de Uso

En este apartado se detalla toda la información referente a casa caso de uso detectado mediante tablas que tendrán los siguientes campos:

- **Actores que intervienen**
- **Propósito del caso de uso**
- **Tipo de Caso de Uso**

- **Referencia a los requisitos funcionales**
- **Prerequisitos que deben cumplirse**
- **Escenario principal o curso típico de eventos**
- **Alternativas o curso alternativo de eventos**

CASO DE USO 01: Gestionar sinónimos de clases	
ACTORES	Usuario de anotaciones
PROPÓSITO	Cada clase tiene asignada una lista de sinónimos obtenidos de Wordnet. Sobre esta lista el usuario podrá: • Añadir sinónimos • Eliminar sinónimos
TIPO	Secundaria, No esencial
REFERENCIAS	REQ.#14, REQ.#15
PREREQUISITOS	1. El usuario ha de hacer click sobre alguna clase para poder así cargar desde el diccionario sus correspondientes sinónimos. 2. Para que el usuario pueda eliminar un sinónimo debe de haber alguno en la lista.
ESCENARIO PRINCIPAL	1. El usuario conoce un término que no se encuentra entre los sinónimos ofrecidos y cree conveniente añadirlo para mejorar así el resultado obtenido en la parte conceptual 2. Alguno de los sinónimos (si existieran) genera resultados inadecuados en la vista conceptual y se elimina para poder así mejorar el resultado.
ALTERNATIVAS	No aplicable

Tabla 3: Caso de Uso: Gestionar sinónimos de Clases

CASO DE USO 02: Acciones sobre Clases y Relaciones	
ACTORES	Usuario de anotaciones
PROPÓSITO	El usuario podrá guiar a la aplicación para que esta le muestra vistas físicas adaptadas a sus necesidades y preferencias. Para ello estas son las siguiente acciones que se podrán realizar: • Descartar Clases y Relaciones • Generalizar Clases • Especificar Clases • Hacer que una Clase sea identificadoras • Añadir restricciones sobre las Clases

TIPO	Primario, Esencial
REFERENCIAS	REQ.#16, REQ.#17, REQ.#18, REQ.#19, REQ.#20
PREREQUISITOS	1. El usuario deberá hacer *click* sobre la Clase o la Relación sobre la que quiera realizar una acción. 2. Para poder realizar algunas de las acciones sobre las clases, estas tienen que cumplir los criterios impuestos por el modelo físico.
ESCENARIO PRINCIPAL	1. El usuario necesita determinar con que Clases y Relaciones quiere trabajar y cuales quiere descartar para no trabajar mas con ellas. 2. El usuario quiere generalizar las anotaciones en base a una clase seleccionada 3. El usuario quiere extender su anotación en base a una clase seleccionada. 4. El usuario especifica que Clases van a ser identificadoras 5. El usuario necesita añadir restricciones sobre las clases
ALTERNATIVAS	No aplicable

Tabla 4: Caso de Uso: Acciones sobre Clases y Relaciones

CASO DE USO 03: Acciones sobre Vistas	
ACTORES	Usuario de anotaciones
PROPÓSITO	El usuario puede tomar decisiones sobre las vistas físicas mostradas por el sistema. Para ello las acciones debe de realizar son las siguientes: • Aceptar Vista • Descartar Vista • Generalizar Vista • Especificar Vista
TIPO	Primario, Esencial
REFERENCIAS	REQ.#13, REQ.#14, REQ.#15
PREREQUISITOS	1. Para generalizar la rama principal de la vista tienen que cumplirse los criterios impuestos por el modelo físico. 2. Para poder especificar la rama principal de la vista la última clase de este tiene que ser especificable. Criterio que también viene impuesto por el modelo físico.
ESCENARIO PRINCIPAL	1. El usuario necesita extender o acortar la rama principal de la vista en función de sus preferencias y necesidades. 2. El usuario entiende que las anotaciones mostradas en lenguaje natural cumplen con sus necesidad y decide añadirlas al resultado final. 3. El usuario no está satisfecho con el resultado mostrado o la vista mostrada no es con la que desea trabajar por lo que decide descartarla en el resultado final.
ALTERNATIVAS	No aplicable

Tabla 5: Caso de Uso: Acciones sobre Vistas

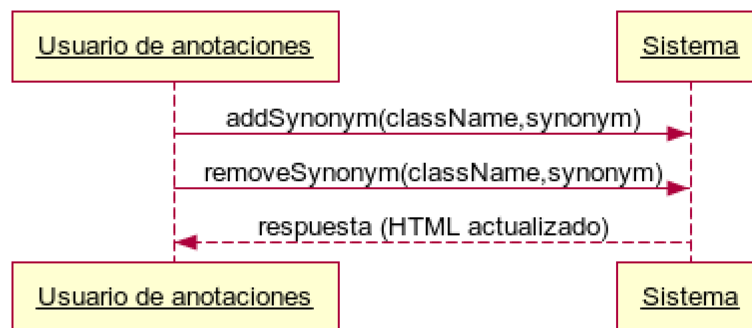
CASO DE USO 04: Gestionar progreso y resultado	
ACTORES	Usuario de anotaciones
PROPÓSITO	El usuario podrá salvar el progreso o terminarlo cuando lo crea procedente. Para ello la aplicación deberá ofrecer las siguiente opciones: • Guardar Progreso • Cargar Progreso • Generar Anotación
TIPO	Secundario, Esencial
REFERENCIAS	REQ.#21, REQ.#22, REQ.#23
PREREQUISITOS	1. Para cargar y retomar un progreso anteriormente ha tenido que ser guardado.
ESCENARIO PRINCIPAL	1. El usuario quiere guardar el trabajo realizado y retomarlo en otro momento. 2. El usuario quiere retomar un trabajo guardado con anterioridad 3. El usuario desea obtener el resultado final
ALTERNATIVAS	No aplicable

Tabla 6: Caso de Uso: Gestionar progreso y resultado

5.3.4 Diagramas de Secuencia del Sistema

En este apartado se van a mostrar los diagramas de secuencia pertenecientes a los casos de uso. Estos diagramas tienen como fin mostrar de forma gráfica la interacción entre Actor y Sistema mediante mensajes, llamadas u operaciones. Además describen más a fondo conceptos de programación como métodos y clases.

5.3.4.1 Diagrama de secuencia para el caso de uso "Gestionar sinónimos de clases"



El caso de uso comienza cuando el usuario quiere mejorar los resultados que la herramienta le está ofreciendo. Para ello podrá realizar las siguientes acciones:

- Añadir sinónimos o términos equivalentes a la clase de forma manual.
- Eliminar un sinónimo o término equivalente a la clase de forma manual

5.3.4.2 Diagrama de secuencia del caso de uso "Acciones sobre clases y relaciones"

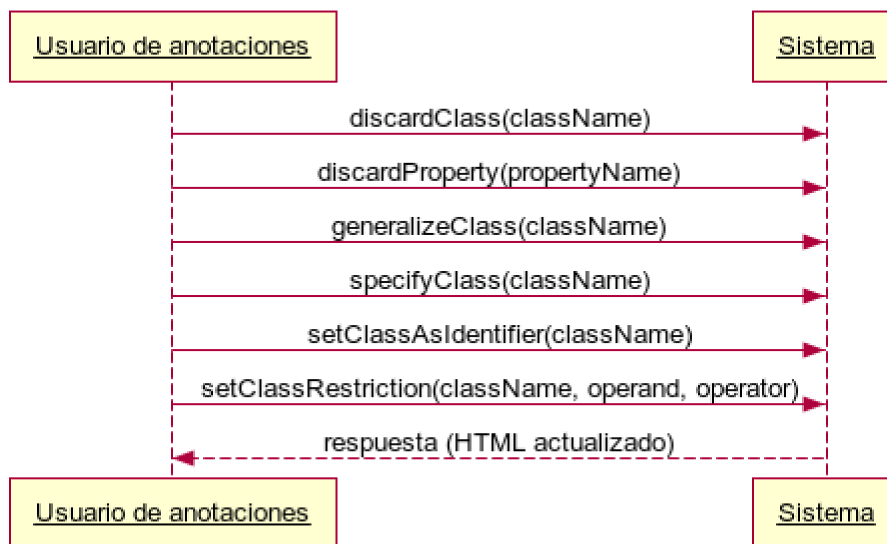


Figura 18: Diagrama de secuencia de Acciones sobre Clases y

El caso de uso comenzará cuando el usuario haga *click* sobre una Clase o Relación y realice una acción sobre dicho objeto. El resultado devuelto por el sistema siempre es una cadena de caracteres que contiene código HTML. Esta cadena cambiara en función de la acción que se realice. En los casos en que se genere una nueva vista física, el HTML devuelto por el sistema también contendrá la nueva vista conceptual generada. Al añadir restricciones o hacer una clase identificadora, solo se recibirá el HTML de la vista física.

5.3.4.3 Diagrama de secuencia para el caso de uso "Acciones sobre vistas"

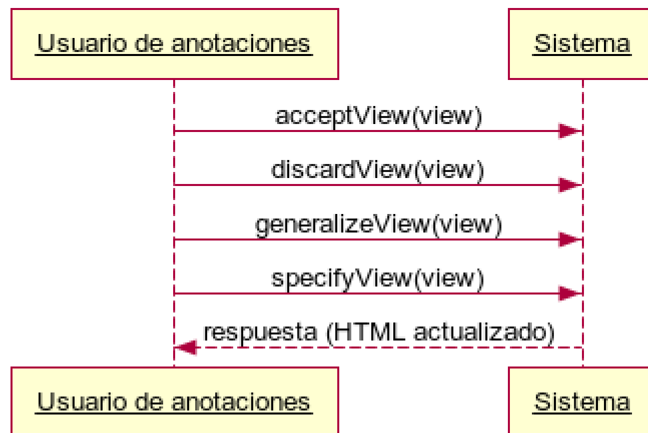


Figura 19: Diagrama de secuencia de Acciones sobre Vistas

El caso de uso comienza cuando un usuario decide realizar una tarea concreta sobre la vista física y su parte conceptual. Todos los casos aquí implican que el HTML devuelto por el sistema contenga tanto la nueva vista física generada como su “equivalente conceptual”. Tanto aceptar la vista como descartarla tienen repercusión en el resultado final.

5.3.4.4 Diagrama de secuencia para el caso de uso “Gestionar progreso y resultado”

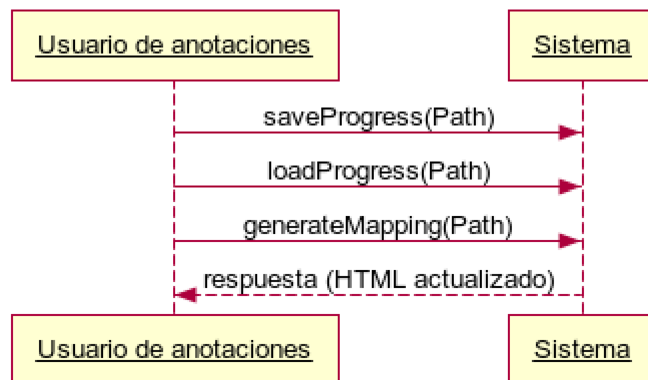


Figura 20: Diagrama de secuencia de Gestionar progreso y resultado

El caso de uso comenzará cuando el usuario estime que quiere realizar una acción sobre el trabajo realizado, ya sea guardar el progreso, cargar el progreso o generar un resultado final. Tanto en el caso de guardar el progreso como en el de generar resultado final, el sistema devolverá un mensaje en HTML informando del resultado. Cargar progreso devolverá el HTML con el progreso del trabajo guardado.

5.3.2 Contratos de las operaciones del Sistema

Las siguientes tablas describen el comportamiento que se espera de las operaciones que han sido mencionadas en el apartado anterior.

Operación: addSynonim(className, synonym)	
Responsabilidades	Añadir un nuevo sinónimo a la lista de sinónimos de una clase
Referencias	Caso de uso "Gestionar sinónimos de clases"
Precondiciones	Ninguna
Postcondiciones	Se añade un nuevo sinónimo a la estructura de datos de sinónimos
Salidas	HTML con la nueva lista de sinónimos
Excepciones	Ninguna

Tabla 7: Contrato de la operación: addSynonym(clasName,synonym)

Operación: removeSynonim(className, synonym)	
Responsabilidades	Eliminar un sinónimo de la lista de sinónimos de una clase
Referencias	Caso de uso "Gestionar sinónimos de clases"
Precondiciones	La clase seleccionada ha de tener al menos un sinónimo
Postcondiciones	Se elimina el sinónimo de la estructura de sínonimos
Salidas	HTML con la nueva lista de sinónimos (Si hubiera alguno)
Excepciones	No existe el sinónimo en la estructura

Tabla 8: Contrato de la operación: removeSynonym(clasName,synonym)

Operación: discardClass(className)

Responsabilidades	Hacer que una clase no vuelva a ser incluida en las vistas físicas futuras.
Referencias	Caso de uso “Acciones sobre clases y relaciones”
Precondiciones	La clase ha de estar incluida en la vista física actual
Postcondiciones	La clase se elimina de la estructura de clases descartadas
Salidas	Siguiente vista física
Excepciones	Ninguna

Tabla 9: Contrato de la operación: discardClass(className)

Operación: discardProperty(propertyName)	
Responsabilidades	Hacer que una property no sea incluida en las vistas físicas futuras
Referencias	Caso de uso “Acciones sobre clases y relaciones”
Precondiciones	La property ha de estar incluida en la vista física actual
Postcondiciones	La property se elimina de la estructura de properties
Salidas	Siguiente vista física
Excepciones	Ninguna

Tabla 10: Contrato de la operación: discardProperty(propertyName)

Operación: generalizeClass(className)	
Responsabilidades	Generalizar la vista física en base a la clase seleccionada.
Referencias	Caso de uso “Acciones sobre clases y relaciones”
Precondiciones	La clase seleccionada ha de estar incluida en la vista física actual y cumplir con las condiciones del modelo físico.
Postcondiciones	Ninguna
Salidas	Vista generalizada en base a la clase
Excepciones	Ninguna

Tabla 11: Contrato de la operación: generalizeClass(className)

Operación: specifyClass(className)	
Responsabilidades	Crear una nueva rama a partir de la clase seleccionada
Referencias	Caso de uso “Acciones sobre clases y relaciones”
Precondiciones	La clase ha de estar incluida en la vista física actual y cumplir con las condiciones del modelo físico.
Postcondiciones	Ninguna
Salidas	Vista con la nueva rama
Excepciones	Ninguna

Tabla 12: Contrato de la operación: specifyClass(className)

Operación: setClassAsIdentifier(className)	
Responsabilidades	Hacer que una clase sea identificadora
Referencias	Caso de uso “Acciones sobre clases y relaciones”
Precondiciones	La clase ha de estar incluida en la vista física actual
Postcondiciones	La clase se etiqueta como identificadora en la estructura
Salidas	Ninguna
Excepciones	Ninguna

Tabla 13: Contrato de la operación: setClassAsIdentifier(className)

Operación: setClassRestriction(className, operand, operator)	
Responsabilidades	Añadir una restricción sobre la clase seleccionada
Referencias	Caso de uso “Acciones sobre clases y relaciones”
Precondiciones	La clase ha de estar incluida en la vista física actual
Postcondiciones	La restricción se añade a la estructura de datos
Salidas	Ninguna
Excepciones	Ninguna

Tabla 14: Contrato de la operación: setClassRestriction(className, operand, operator)

Operación: acceptView(view)	
Responsabilidades	Incluir las vistas en el resultado final
Referencias	Caso de uso “Acciones sobre vistas”
Precondiciones	Ninguna
Postcondiciones	Tanto la vista física como la conceptual se añaden a la estructura de datos que servirá para generar el resultado final.
Salidas	Siguiente Vista
Excepciones	Ninguna

Tabla 15: Contrato de la operación: acceptView(view)

Operación: discardView(view)	
Responsabilidades	Excluir las vistas del resultado final
Referencias	Caso de uso “Acciones sobre vistas”
Precondiciones	Ninguna
Postcondiciones	Las vistas se desechan
Salidas	Siguiente Vista
Excepciones	Ninguna

Tabla 16: Contrato de la operación: discardView(view)

Operación: generalizeView(view)	
Responsabilidades	Se generaliza la vista física
Referencias	Caso de uso “Acciones sobre vistas”
Precondiciones	El modelo de datos físico ha de permitir que la vista sea generalizable
Postcondiciones	Ninguna
Salidas	Vista generalizada
Excepciones	Ninguna

Tabla 17: Contrato de la operación: generalizeView(view)

Operación: specifyView(view)	
Responsabilidades	Extender la vista a partir de la última clase de la rama principal de la vista
Referencias	Caso de uso “Acciones sobre vistas”
Precondiciones	La última clase de la rama principal se debe ser especificable
Postcondiciones	Ninguna
Salidas	Vista especificada
Excepciones	Ninguna

Tabla 18: Contrato de la operación: specifyView(view)

Operación: saveProgress(Path)	
Responsabilidades	Se salva el progreso del trabajo y se almacena en un fichero de la ruta especificada
Referencias	Caso de uso “Gestión de progreso y resultado”
Precondiciones	Ninguna
Postcondiciones	Se genera un fichero XML con las acciones realizadas
Salidas	Ninguna
Excepciones	La ruta no es válida

Tabla 19: Contrato de la operación: saveProgress(Path)

Operación: loadProgress(Path)	
Responsabilidades	Se carga el progreso guardado anteriormente en un fichero
Referencias	Caso de uso “Gestión de progreso y resultado”
Precondiciones	El fichero de progreso debe existir
Postcondiciones	Se ejecutan todas las acciones realizadas en el progreso que se salvo con anterioridad
Salidas	Ultima vista mostrada despues de salvar el progreso
Excepciones	El fichero no existe

Tabla 20: Contrato de la operación: loadProgress(Path)

Operación: generateMapping(Path)	
Responsabilidades	Se genera el fichero final con las anotaciones
Referencias	Caso de uso “Gestión de progreso y resultado”
Precondiciones	Ninguna
Postcondiciones	Se genera un fichero XML con las anotaciones realizadas
Salidas	Ninguna
Excepciones	La ruta no es válida

Tabla 21: Contrato de la operación: generateMapping(Path)

CAPÍTULO 6

DISEÑO E IMPLEMENTACIÓN DE LA HERRAMIENTA

En este capítulo se describe detalladamente el diseño y la implementación de la herramienta. Al ser una aplicación que engloba el proyecto p-medicine, las tecnologías de desarrollo utilizadas son las mismas. A continuación se justifica este hecho.

6.1 Arquitectura de la herramienta

El desarrollo de esta herramienta se compone de dos partes: La parte del usuario (Interfaz Web) y La parte del servidor (Aplicación Java).

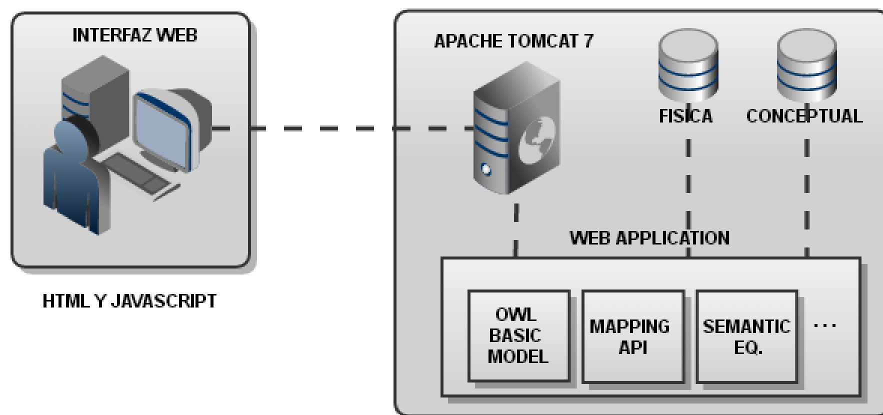


Figura 21: Arquitectura de la herramienta

La Figura 21 muestra gráficamente la arquitectura típica de una aplicación web, en este caso desarrollada en Java y desplegada en un servidor Tomcat. La comunicación entre la interfaz de usuario y la parte programática se realiza mediante los protocolos de comunicación por internet. Esencialmente, todo el trabajo lo realiza el servidor y los mensajes que se le envían al usuario son cadenas de HTML que servirán para mostrar los resultados en la página web.

6.2 Interfaz Web

La interfaz tiene una sola página principal donde se van cargando los diferentes módulos en función de las acciones que se vayan realizando. Utilizando Jquery se carga la página por partes y el usuario no tendrá la sensación de estar cargando una nueva página por cada acción realizada. Esto mejora la usabilidad de la aplicación, uno de los requisitos no funcionales descritos en el apartado de "Análisis del Sistema". A continuación se listan las páginas web de las que dispone el interfaz.

index.jsp

Página principal de la herramienta.

classActions.jsp

Página a la que el cliente llama cuando se realiza una acción sobre una clase.

generateMapping.jsp

Página a la que el cliente llama cuando quiere generar el resultado final.

propertyActions.jsp

Página a la que el cliente llama cuando se realiza una acción sobre un relación.

serializeActions.jsp

Página a la que el cliente llama para salvar o cargar el progreso del trabajo.

viewActions.jsp

Página a la que el cliente llama cuando se realiza una acción sobre la vista.

La página principal es *index.jsp* que contiene todos los contenedores donde se cargarán el resto de páginas web. Cabe matizar que la extensión es *jsp* y no *html* ya que por convenio se aplica esta primera cuando entre el código HTML existe parte de código Java incrustado.

6.3 Aplicación en el servidor

La aplicación en la parte del servidor, desarrollada en Java, hace uso de APIs externas desarrolladas por el GIB. A continuación se justifica el uso de cada una de ellas:

6.3.1 Clases

Las clases Java que aparecen en este módulo son:

6.3.1.1 ViewContainer

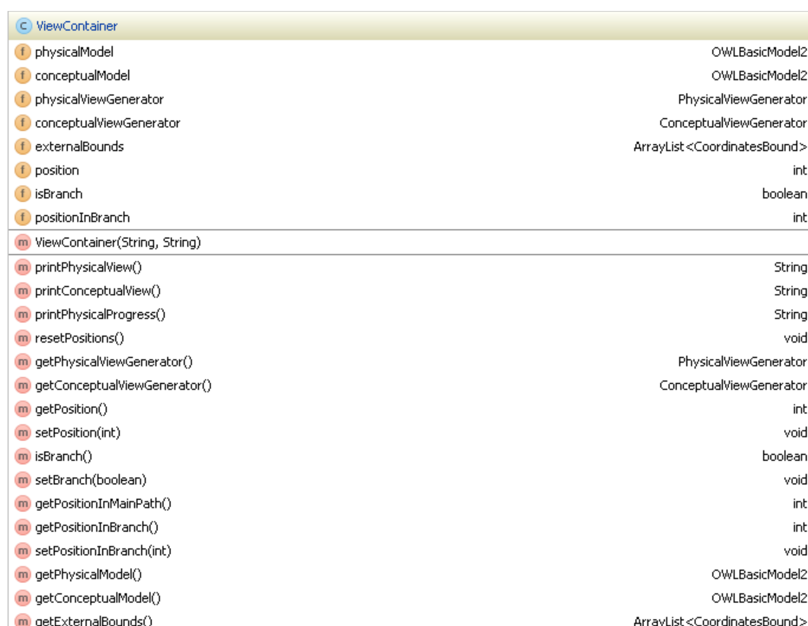


Figura 22: Clase ViewContainer

Esta clase puede verse como el contenedor de todos los objetos Java que se van a utilizar en la herramienta: OWLBasicModel2, PhysicalViewGenerator, ConceptualViewGenerator y ExternalBounds. Este clase ofrece métodos Get y Set para los objectos mencionados así como métodos para imprimir las vistas generadas en HTML.

6.3.1.2 SimilarityStructure

SimilarityStructure	
physicalModel	OWLBasicModel2
conceptualModel	OWLBasicModel2
classSimilarities	Map<String, Map<Integer, ArrayList<String>>>
propertySimilarities	Map<String, Map<Integer, ArrayList<String>>>
synonyms	Map<String, LinkedList<String>>
methods	Methods
SimilarityStructure(OWLBasicModel2, OWLBasicModel2)	
addSimilarity(String, similarityType, ArrayList<String>)	Map<Integer, ArrayList<String>>
getLabel(String, similarityType, boolean)	String
addSynonyms(String)	LinkedList<String>
putSynonym(String, String)	void
removeSynonym(String, String)	void
getPropertySimilarities()	Map<String, Map<Integer, ArrayList<String>>>
getClassSimilarities()	Map<String, Map<Integer, ArrayList<String>>>
getSynonyms()	Map<String, LinkedList<String>>

Figura 23: Clase SimilarityStructure

Esta clase es una estructura de similitud entre los modelos físico y conceptual. Los atributos son Maps en los que se almacenan los términos físicos junto a un listado de posibles equivalencias en la parte conceptual. Otro atributo llamado synonyms almacena sinónimos obtenidos de WordNet de cada término en la parte física.

6.3.1.3 PrintText

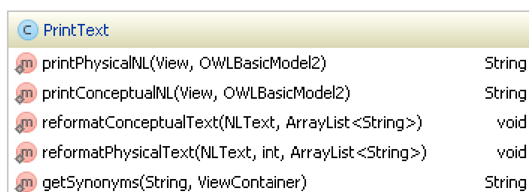


Figura 24: Clase PrintText

Esta clase ofrece métodos para formatear a HTML el texto en lenguaje natural generado por la API MappingAPI2. También tiene implementado un método para devolver el listado de sinónimos de un término.

6.3.1.4 ActionsClass

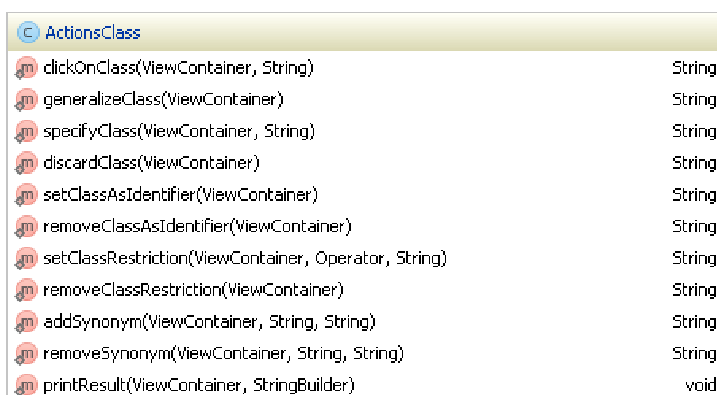


Figura 25: Clase ActionsClass

Esta clase tiene implementados los métodos para todas las acciones que se pueden realizar al hacer *click* sobre una clase.

6.3.1.5 ActionsProperties

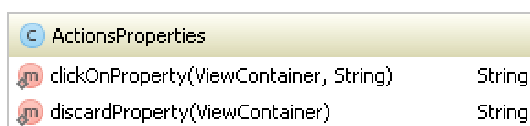


Figura 26: Clase ActionsProperties

Esta clase tiene implementados los métodos para la acciones que se pueden

realizar al hacer *click* sobre una relación.

6.3.1.6 Buttons

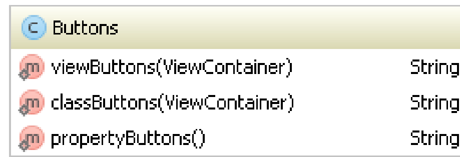


Figura 27: Clase Buttons

Esta clase tiene implementados métodos para devolver código HTML de los botones para realizar acciones sobre vistas, clases o relaciones.

6.3.1.7 SerializeBox

Figura 28: Clase SerializeBox

Esta clase tiene implementados métodos para devolver el código HTML del formulario para cargar o salvar el progreso de trabajo.

6.3.1.8 ConceptualViewGenerator

Figura 29: Clase ConceptualViewGenerator

Esta clase es la responsable de generar las vistas conceptuales más verosímiles. Cada vez que se genera una vista física en la aplicación, este objeto generará su equivalente en la parte conceptual. Para ello se vale del atributo similarityStructure.

CAPÍTULO 7
EXPERIMENTOS Y RESULTADOS

En este capítulo se describen las pruebas realizadas y se justifica el uso de los algoritmos. Los RDF con los que se han realizado estas pruebas son parte de las bases de datos clínicas usadas en p-medicine. Además para la parte conceptual se ha usado la versión más actualizada de HDOT, ya que es la ontología utilizada por el proyecto p-medicine, en el que este trabajo se engloba.

Los experimentos que se van a realizar servirán para determinar la tasa de aciertos y de resultados válidos. Para ello haremos uso de 2 modelos físicos diferentes tomando siempre HDOT como modelo conceptual. A continuación se listan los experimentos que se van a realizar:

- **Experimento 1 (Tasa de aciertos sin sinónimos)**

Análisis de la tasa de aciertos sin uso de sinónimos. Se analizarán los resultados para las dos bases de datos físicas.

- **Experimento 2 (Tasa de aciertos con sinónimos)**

Análisis de la tasa de aciertos usando la estructura de sinónimos con el fin de mejorar los resultados obtenidos en el experimento anterior.

- **Experimento 3 (Umbrales Damerau-Levenshtein)**

Análisis de umbrales para la distancia Damerau-Levenshtein de manera que el algoritmo sea más eficiente para la que se medirán los tiempos.

Las bases de datos físicas que se emplean para realizar los experimentos son subconjuntos de bases de datos clínicas mas extensas que son empleadas en el proyecto p-medicine. Ambas bases de datos contienen información relacionada con el cáncer.

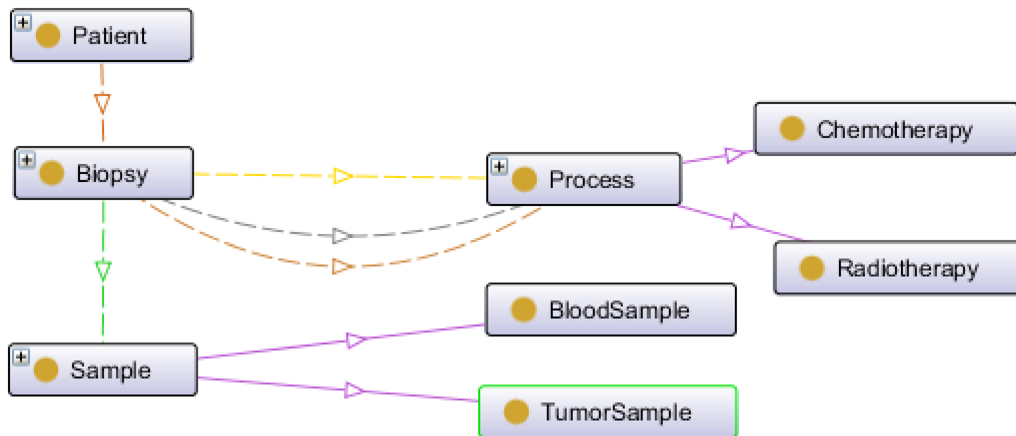


Figura 30: Representación gráfica de la base de datos ClinicalDB1

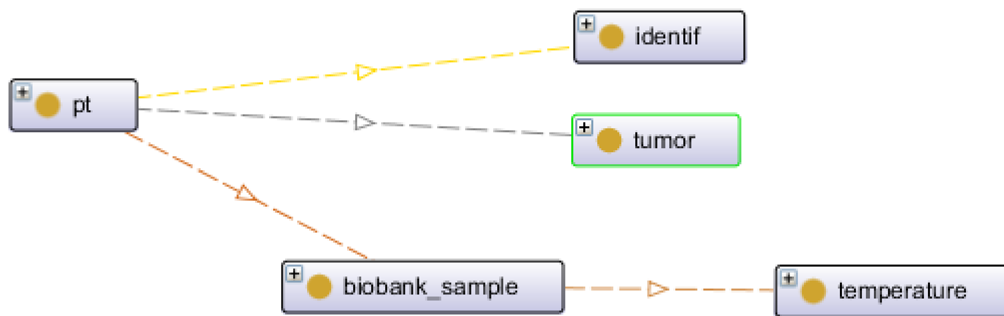


Figura 31: Representación gráfica de la base de datos ClinicalDB2

Base de datos	ClinicalDB1	ClinicalDB2
Número de clases	8	5
Número de relaciones	5	4

Tabla 22: Características de las bases de datos físicas

7.1 Experimentos

7.1.1 Tasa de aciertos sin sinónimos

En este experimento se considerarán aciertos todas aquellas clases que tienen una distancia Damerau-Levenshtein igual a 0, lo que significa que el propio término existe como tal en HDOT. Se considerará como resultado satisfactorio además del acierto, todo término que independientemente de su distancia el experto considere que es equivalente.

ClinicalDB1		
Clase Física	Clase Conceptual	Distancia Damerau-Levenshtein
Biopsy	BioXSD	2
Patient	Patient	0
Process	Process	0
Chemotherapy	Chemotherapy	0
Radiotherapy	Chemotherapy	5
Sample	Simple	1
BloodSample	Biobank Sample	6
TumorSample	Numeral	6

Tabla 23: Análisis de tasa de aciertos sin sinónimos para ClinicalDB1

ClinicalDB2		
Clase Física	Clase Conceptual	Distancia Damerau-Levenshtein
Biobank_sample	Biobank sample	1
identif	Identifier	3
pt	Pt2	1
temperature	Temperature	0

tumor	Taxon	3

Tabla 24: Análisis de tasa de aciertos sin sinónimos para ClinicalDB2

En el experimento realizado con la base de datos "ClinicalDB1", 3 de las 8 clases coinciden exactamente con alguna de las clases en la parte conceptual. El resto de clases, aun teniendo distancias relativamente cortas, han devuelto resultados incorrectos.

En el experimento realizado con la base de datos "ClinicalDB2" solo la clase "temperature" existe como tal en HDOT. No obstante, para las clases "Biobank_sample" e "identif" con distancias 1 y 3 respectivamente se han obtenido resultados satisfactorios.

Resultados obtenidos		
Base de datos	Aciertos	Resultado Satisfactorio
ClinicalDB1	37,5%	37,5%
ClinicalDB2	20%	60%

Tabla 25: Resultados de tasa de aciertos sin sinónimos

7.1.2 Tasa de aciertos con sinónimos

En este experimento se considerarán aciertos todas aquellas clases que tienen una distancia Damerau-Levenshtein igual a 0 o que alguno de sus sinónimos tenga una distancia equivalente a 0. Se considerará resultado satisfactorio además del acierto, toda clase que independientemente de su distancia el experto considere que es equivalente.

ClinicalDB1		
Clase Física	Clase Conceptual	Distancia Damerau-Levenshtein
Biopsy	BioXSD	2
Patient	Patient	0
Process	Process	0
Chemotherapy	Chemotherapy	0

Radiotherapy	Rumination	5
Sample	Simple	1
BloodSample	Biobank Sample	6
TumorSample	Numeral	6

Tabla 26: Análisis de tasa de aciertos con sinónimos para ClinicalDB1

ClinicalDB2		
Clase Física	Clase Conceptual	Distancia Damerau-Levenshtein
Biobank_sample	Biobank sample	1
identif	Identifier	3
pt	Pt2	1
temperature	Temperature	0
tumor	Neoplasm	3

Tabla 27: Análisis de tasa de aciertos con sinónimos para ClinicalDB2

En el experimento realizado con la base de datos “ClinicalDB1” no se han mejorado los resultados con respecto al primer experimento. Esto puede ser debido a que los nombres de algunas clases están compuestos por dos términos diferentes, como “BloodSample” o “TumorSample”. El diccionario no es capaz de encontrar sinónimos para estas clases. Esto se puede solucionar si el usuario añade sinónimos manualmente.

En el experimento realizado con la base de datos “ClinicalDB2” sin embargo si ha mejorado los resultados. La clase tumor, que no tenía un *match* en el experimento anterior, ahora la tiene gracias a su sinónimo “neoplasm”, que si se encuentra en HDOT.

Resultados obtenidos		
Base de datos	Aciertos	Resultado Satisfactorio
ClinicalDB1	37,5%	37,5%
ClinicalDB2	40%	80%

Tabla 28: Resultados de tasa de aciertos con sinónimos

7.1.3 Umbrales Damerau-Levenshtein

El propósito de este experimento es medir tiempos y recursos cambiando los umbrales para la distancia de este algoritmo. Si se acota mucho la distancia se corre el riesgo de no encontrar ningún término similar en HDOT, por lo que no sería factible generar una vista conceptual. A medida que el umbral crece la posibilidad de generar una vista conceptual incrementa, sin embargo esto puede llevar a resultados insatisfactorios.

Las siguiente tabla muestra la cantidad de similitudes que se guardan en la estructura de similitud en función del umbral establecido.

Cantidad de términos generados en función del umbral Damerau-Levenshtein		
Umbral	**ClinicalDB1**	**ClinicalDB2**
50	425	283
25	200	133
10	65	49
5	23	15

Tabla 29: Cantidad de similitudes en función del umbral Damerau-Levenshtein

La siguiente tabla muestra el tiempo que tarda el sistema en generar la estructura de similitudes. Los datos que se muestran han sido obtenidos de un ordenador de sobremesa Core 2 Duo y 2048Mb de memoria RAM. La medida se ha tomado en segundos y se ha sacado la media de tres mediciones diferentes para cada caso.

Medición en segundos para la generación de la estructura de similitudes		
Umbral	**ClinicalDB1**	**ClinicalDB2**
50	5,571	1,708
25	5,502	1,562
10	5,409	1,505
5	5,401	1,503

Tabla 30: Mediciones de tiempo para la generación de la estructura de similitudes

De los datos obtenidos se puede determinar que el tiempo empleado para la creación de la estructura de similitud no incrementa tanto en función del umbral en la distancia Damerau-Levenshtein como en la cantidad de clases que contiene la base de datos física. Sin embargo, tener tablas con muchas entradas puede entorpecer el rendimiento, ya que es una estructura a la que se accede con mucha frecuencia.

Reducir el umbral a 5 ha generado buenos resultados en “ClinicalDB2” obteniendo siempre un término similar en la parte conceptual. Sin embargo este umbral es muy corto para “ClinicalDB1” donde al menos dos clases no tenían similitud en la parte conceptual, dato que se puede contrastar observando los resultados en el experimento 1. Por lo tanto podemos estimar que una distancia Damerau-Levenshtein menor o igual a 10 es el umbral más eficaz.

7.2 Caso Práctico

En este apartado se mostrará el funcionamiento de la herramienta empleando la base de datos ClinicalDB2. En el siguiente “paso a paso” se mostrará tanto el funcionamiento como los resultados que se generan con la herramienta *Ontology Annotator Assistant.*

Figura 32: Punto de partida en la base de datos ClinicalDB2

Tal como se ha descrito en la sección de métodos, el algoritmo para la creación de paths posibles tendrá como punto de partida un nodo raíz. En el caso de ClinicalDB2 solo hay un nodo raíz: “pt”.

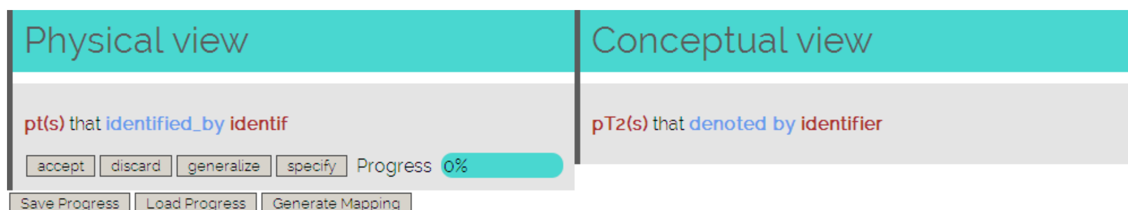


Figura 33: Punto de partida de ClinicalDB2

En la Figura 33 se muestra el punto de partida del proceso, donde partiendo de la clase física “pt” se empieza a recorrer desde la relación “identified_by”. El alineamiento entre la clase física “pt” y la clase conceptual “pT2” no es correcto, ya que aunque parecen términos similares tienen significados diferentes. El usuario sabe sin embargo que “pt” es una simplificación para el término paciente, por lo que hará *click* sobre la clase y añadirá la palabra “patient” como sinónimo de esta clase tal como se muestra en la Figura 34.

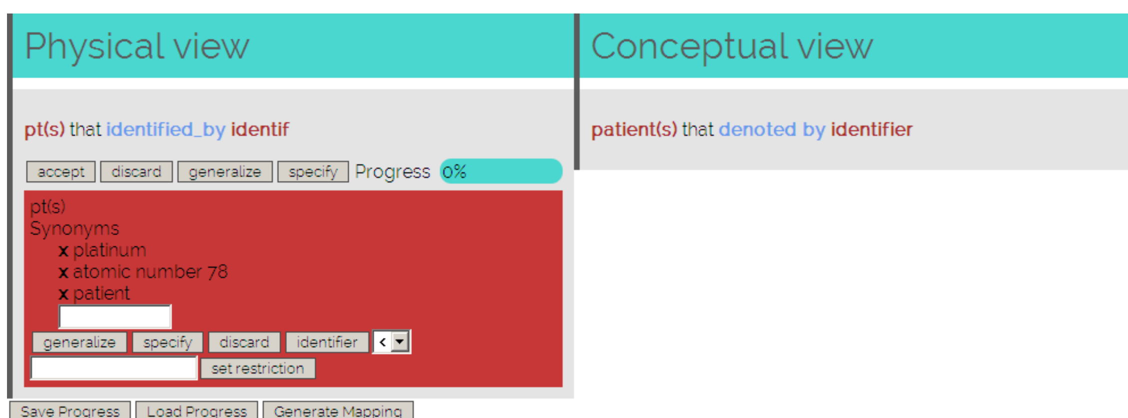


Figura 34: El usuario añade la palabra “patient” como sinónimo de “pt”

Cuando se añade el sinónimo, automáticamente se actualiza la vista conceptual

con el nuevo resultado. Como “patient” es un término que existe en HDOT se sustituye por “pT2”. De esta forma el resultado es correcto y el usuario aceptará la vista para añadirla al resultado final y el algoritmo seguirá recorriendo la base de datos física.

Figura 35: Tras aceptar la vista se genera la siguiente

Figura 36: El diccionario contiene "neoplasm" como sinónimo de "tumor"

“Tumor” no es una clase contenida en HDOT y su término más similar es “Taxon”. Sin embargo, Wordnet contiene la palabra “Neoplasm” (clase existente en HDOT) como sinónimo de tumor y por lo tanto el resultado mostrado también es aceptable.

La siguiente vista que se genera termina en la clase “biobank_sample”, clase que es especificable ya que se puede recorrer la relación “has_quality”. La vista se alargaría pasando a tener longitud 2, tal como se muestra en la Figura 37.

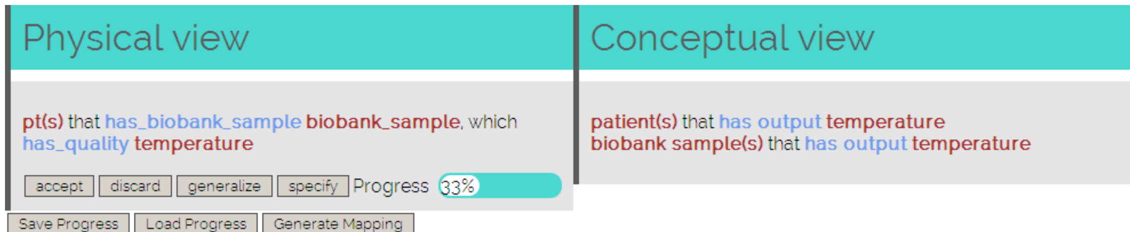


Figura 37: Se especifica la vista a partir de la clase biobank_sample

Figura 38: Al generalizar la vista se parte desde biobank_sample

Si el usuario quisiera simplificar la vista que ha sido especificada, podría en este punto generalizar de manera que la vista volvería a tener longitud 1 y partiría de la clase “biobank_sample”. Una vez aceptada esta última vista, el progreso llegaría al 100% y el trabajo estaría finalizado.

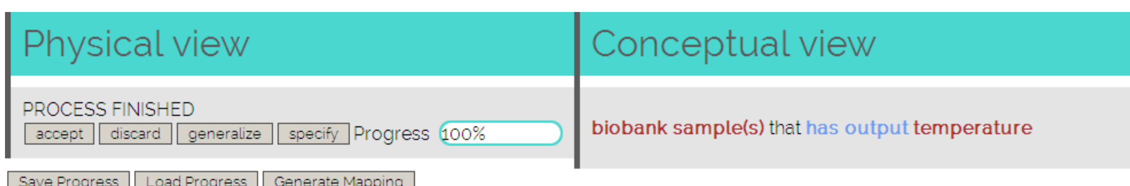


Figura 39: Al llegar al 100% del progreso el trabajo habrá terminado

CAPÍTULO 8
CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo final se expondrán las conclusiones a las que se ha llegado tras la realización de este trabajo de fin de grado y se expondrán propuestas de mejora para el sistema desarrollado.

8.1. Conclusiones

El objetivo planteado en este trabajo de fin de grado era diseñar e implementar una aplicación que asista a un usuario en la tarea de generar anotaciones basadas en vistas para fuentes RDF. Esta herramienta está pensada para ser integrada dentro del proyecto p-medicine con la intención de facilitar la tarea a todos los usuarios que encuentran dificultades a la hora de generar anotaciones con el *Ontology Annotator*. Tras el estudio realizado durante el desarrollo de este trabajo se ha llegado a las siguientes conclusiones:

- **Búsqueda de similitudes término a término**

En este trabajo se han intentado conseguir los mejores resultados mediante el uso del algoritmo de Damerau-Levenshtein y sinónimos extraídos de un diccionario. Los resultados son buenos cuando las clases tienen nombres simples, en cambio empeoran drásticamente cuando estos están compuestos por varias palabras. Este problema se puede encontrar tanto en la parte física como en la conceptual.

Se podrían mejorar sensiblemente los resultados del sistema si se hiciera un estudio previo de cada clase. Algunas de las clases están compuestas por dos palabras separadas

por una “barra baja” (biobank_sample), en otras ocasiones la separación de palabras se da con la introducción de letras mayúsculas (tumorSample). Si se hiciera un análisis previo y se estudiara la similitud en base a dos términos, los resultados mejorarían.

- **Similitudes entre relaciones**

Encontrar similitudes entre relaciones es más complejo si cabe que con las clases, ya que generalmente los nombres son verbos o pequeñas frases separadas por “barra baja” o incluso por espacios. Generalmente las relaciones tienen la forma “has *something*” por lo que pierde todo sentido intentar buscar una similitud término a término. Un caso flagrante es el de la relación “proper part of continuant at all times”.

En este trabajo no se ha profundizado mucho en la mejora de esta característica en parte porque el alineamiento entre clases es más crítico que entre relaciones. Además las relaciones implementadas en HDOT son muy poco restrictivas y en ocasiones no tienen rango, lo que las convierte en una relación “a todos”. Por esta razón sería muy complejo dar con una relación acertada.

- **Creación de vistas en la base de datos física**

La base de datos de la parte física no debería ser muy extensa ni contener ciclos. En el caso de que el tamaño de la base de datos física fuera considerablemente grande el proceso de recorrerla entera sería muy largo. Conviene que la parte física sea un subconjunto bien acotado de una base de datos, de manera que la cantidad de vistas generables sea factible para el usuario.

- **Creación de triples conceptuales con clases “todos a todos”**

El algoritmo automático para la generación de vistas verosímiles genera una lista de triples que une todas las clases con todas siempre que el modelo conceptual lo permita. De esta manera se abstrae al algoritmo de usar una estructura similar a la parte física, lo que mejora el rendimiento y la eficiencia. Como contrapartida esta cantidad de triples va creciendo de manera exponencial a medida que haya mas clases en la base de datos

física.

Cabe destacar que no existen herramientas con características similares a la desarrollada en este trabajo. Existen algunas herramientas que generan automáticamente anotaciones entre fuentes de datos RDF, pero solo entre términos sueltos, y no entre vistas completas. La complejidad de generar anotaciones para vistas en vez de para términos sueltos ha hecho que la mayor dificultad del trabajo realizado haya residido en diseñar el algoritmo de generación de vistas conceptuales. Los resultados obtenidos en la fase de pruebas son dispares y muestran una gran dependencia en la estructura concreta de las fuentes alineadas. Sin embargo, dichos resultados son a su vez muy prometedores pues logran generar en algunos de los casos anotaciones de gran calidad. Por otra parte, los mecanismos para dirigir parcialmente este proceso, como la posibilidad de que el usuario introduzca sinónimos de los términos físicos, mejora enormemente los resultados y ofrece un gran avance para un problema que es fuertemente dependiente del contexto. La información semántica inherente a las clases de las fuentes de datos físicas a menudo no está disponible para la herramienta, y ha de ser el usuario quien aporte esta información.

Es importante destacar que, en todos los casos, los resultados obtenidos en los experimentos suponen una mejora con respecto al *Ontology Annotator*, el sistema de generación de alineamientos de p-medicine. En primer lugar, el *Ontology Annotator* demostró ser demasiado complejo para el usuario medio, imposibilitando su uso en la mayoría de las situaciones. En segundo lugar, la generación manual de anotaciones resultaba demasiado tediosa y suponía un gran gasto de recursos humanos. Las pruebas realizadas demostraron que el uso del Ontology Annotator Assistant permitía reducir notablemente el tiempo que los usuarios debían dedicar a la generación de ontologías. Si con el *Ontology Annotator* un usuario medio gastaba aproximadamente cuatro minutos por vista, con el *Ontology Annotator Assistant* este tiempo se reducía a alrededor de 20 segundos (el tiempo necesario para verificar que la vista física generada debía ser alineada, y que la correspondiente vista conceptual generada era la esperada). La

herramienta desarrollada cumple por tanto su objetivo principal de facilitar la creación de anotaciones.

La herramienta desarrollada será incluida en la plataforma del proyecto p-medicine, y utilizada en la integración de ensayos clínicos reales en un futuro próximo.

8.2 Líneas futuras

En este apartado se proponen mejoras y posibles desarrollos para un futuro. Hay que tener en cuenta que el alineamiento automático basado en vistas no tiene precedente y se le presupone un largo recorrido de mejora. Las ideas que se exponen a continuación pretenden arrojar ideas para la mejora de resultados.

- **Método alternativo**

Dado que se trabaja con lenguaje natural sería posible hacer un análisis sintáctico (lingüístico) y morfosintáctico de las vistas físicas. Por otro lado sería necesario una estructura de sujetos, predicados y demás recursos lingüísticos generados a partir del modelo conceptual.

Gracias a esto se podría generar una "traducción" de una base de datos a otra análogamente al método que Google usa en su aplicación *Translate* donde hace un parseo del idioma origen y traduce al idioma destino adaptándose a las nuevas normas gramaticales.

De esta forma se desecha la idea de incluir todos los términos y el problema deja de tener un enfoque sintáctico para centrarse más en un enfoque semántico aunque aun sería necesario el uso de sinónimos y algoritmos de similitud entre términos.

- **Mejora en la similitud de relaciones**

Con el fin de dar solución al problema de encontrar similitudes entre relaciones, existen métodos que analizan similitud entre frases en base a la cantidad de palabras equivalentes que contienen. Ya que el problema radica en que las relaciones generalmente no son términos sueltos sino frases cortas.

- **Modificar la vistas generadas en la parte conceptual**

Ontology Annotator Assistant genera las vistas conceptuales automáticamente y estas no son modificables. Si haciendo *click* sobre las clases y relaciones conceptuales se mostraran alternativas adecuadas a estas, el usuario tendría mas flexibilidad a la hora de crear anotaciones y los resultados serían más satisfactorios.

BIBLIOGRAFÍA

[1] Anguita, Alberto; Martin, Luis; Perez-Rey, David; Maojo, Victor, «A Review of Methods and Tools for Database Integration in Biomedicine» 2010

[2] RDF, «Resource Description Framework» 2014. http://www.w3.org/RDF/

[3] W3C, «World Wide Web Consortium» 2014. http://www.w3.org

[4] HDOT: Sanfilippo EM, Schwarz U, Schneider L. «The Health Data Ontology Trunk (HDOT)». Towards an ontological representation of cancer-related knowledge. Proc. IARWISOCI2012, Athens, 2012.

[5] A. Escrich, «Estudio y comparación de métodos de alineamiento de ontologías biomédicas» 2013.

[6] A. Anguita, M. García-Remesal, D. de la Iglesia, N.Graf, V. Maojo, Toward a view-oriented approach for aligning RDF-based, 2014

[7] GNU «GNU General Public License» 2014. http://www.gnu.org/copyleft/gpl.html

[8] Eclipse «Eclipse Kepler», 2014. http://www.eclipse.org/downloads/

[9] IntelliJ «IntelliJ IDEA 13» 2014, http://www.jetbrains.com/idea/download/

[10] Java «Java JDK 7u60 » 2014 http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

[11] JQuery «Jquery 1.9.1 » 2014 , http://jquery.com/download/

[12] N-Triples, «N-Triples» 2014. http://www.w3.org/2001/sw/RDFCore/ntriples/

[13] OWL,«Web Ontology Language» 2014. http://www.w3.org/OWL/

[14] OWL2 ,«Web Ontology Language 2» 2014. http://www.w3.org/TR/owl2-overview/

[15] XML ,«Extensible Markup Language» 2014. http://www.w3.org/XML/

[16] JDOM , 2014. http://www.jdom.org/downloads/

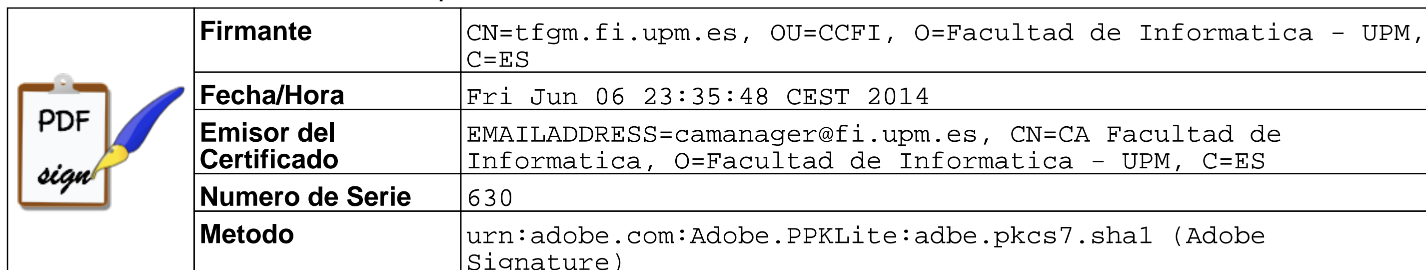
[17] DOM ,«Document Object Model» 2014. http://www.w3.org/DOM/

[18] TOMCAT 7 ,«Apache Tomcat 7» 2014. http://tomcat.apache.org/download-70.cgi

[19] Prótegé ,«Prótegé» 2014. http://protege.stanford.edu/download/protege/4.3/installanywhere/Web_Installers/

[20] JAWS «Java API for WordNet Searching » 2014. http://lyle.smu.edu/~tspell/jaws/index.html

Este documento esta firmado por



	Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Jun 06 23:35:48 CEST 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)